Design, development, and analysis of a compressible fluid dynamics solver capable of exceeding one quadrillion degrees of freedom

Ben Wilfong

Georgia Institute of Technology

October 8th, 2025

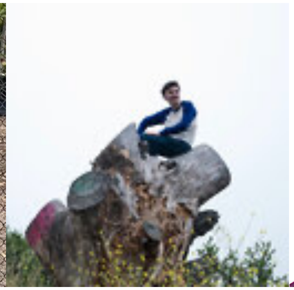
Computational Physics Group

Collaborative effort


Anand
Radhakrishnan


Daniel
Vickers


Tanush
Prathi


Henry
Le Berre


Spencer
Bryngelson

Georgia Tech

Acknowledgements

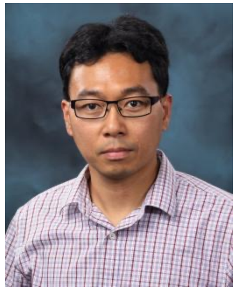
Scott Futral (LLNL)
Rob Noska (HPE)
Michael Sandoval (OLCF)
Mat Colgrove (NVIDIA)

NVIDIA


Nikolaos
Tselepidis


Benedikt
Dorschner

Oak Ridge National Laboratory | Leadership Computing Facility


Reuben
Budiardja

AMD


Brian
Cornille

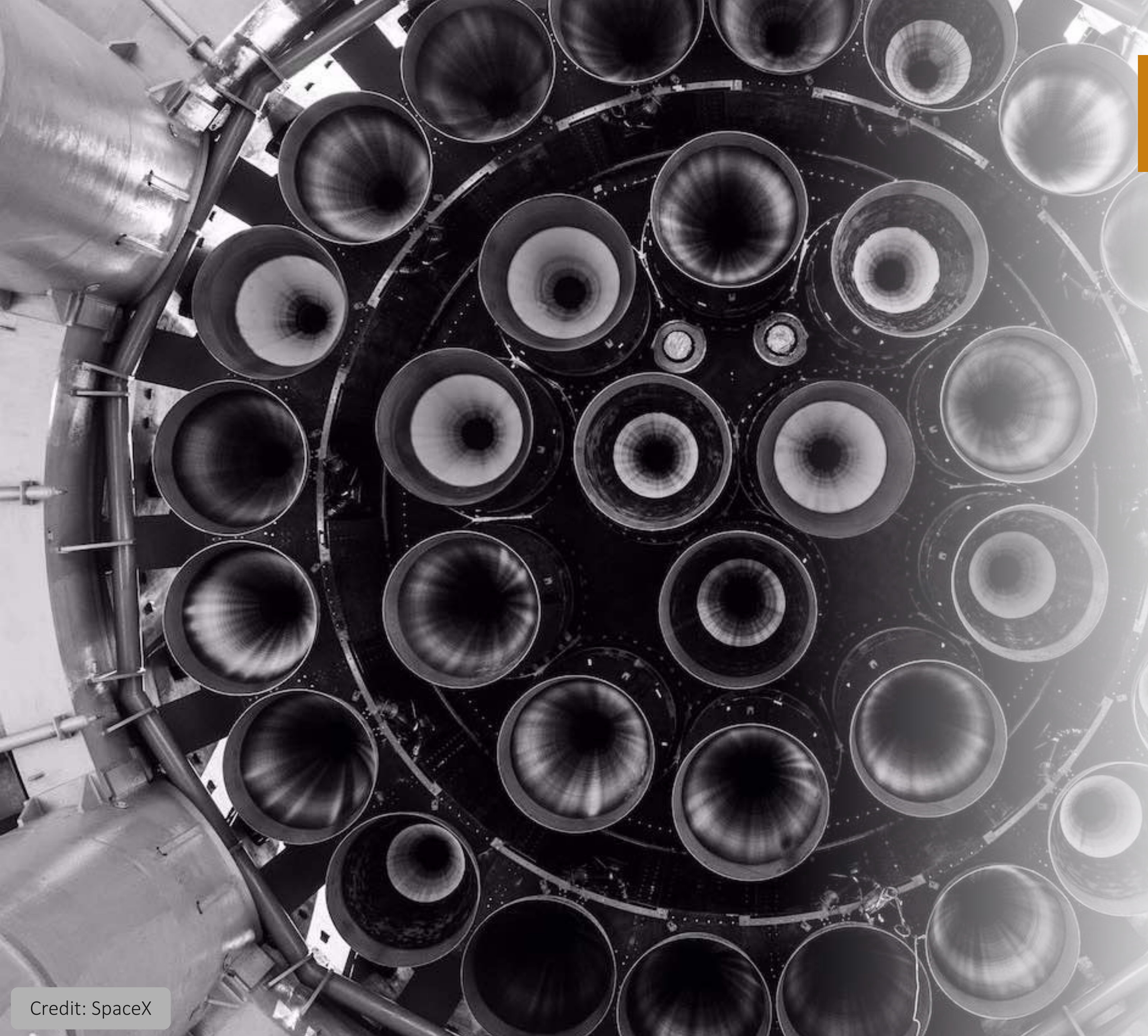
Hewlett Packard Enterprise | CRAY


Stephen
Abbott

NYU | Courant Institute of Mathematical Sciences


Florian
Schäfer

Introduction



Credit: SpaceX

- Large arrays of small rockets are the new standard
- Interactions between nearby rockets produce new problems
- Simulation resolution is limited by available GPU memory
- More resolution = more rockets
- Existing tools for shock-laden flows are poorly conditioned at lower precision
- Lower precision and linear numerics are faster

Timeline of biggest CFD simulations

2013[1]
(GB Winner)

- 10T grid cells on BlueGene/Q
- Enabled by susbstanial CPU memory
- Nonlinear state-of-the-art numerics
- Time to solution ~ 1 month for meaningful time scales

Sep. 2024[2]

- 35T grid cells on OLCF Frontier
- Incompressible flow
- Time-to-solution dominated by all-to-all communication
- Time to solution: hours to days

Oct. 2024[3]

- 10T grid cells on OLCF Frontier
- Nonlinear state-of-the-art numerics
- Time to solution: hours to days
- Compressible

Today

- 200T cells on OLCF Frontier
- 113T cells on LLNL El Cap
- 100T cells on JSC Jupiter
- Compressible flow solver
- Novel numerics
- Time to solution: hours to days

[1] Rossinelli et al. (2013) Proceedings of SC '13
[2] Yeung et al. (2025) CPC
[3] Sathyanarayana et al. (2025) JPDC

Summary of contributions

- Information geometric regularization foregoes nonlinear viscous shock capturing, enabling linear off-the-shelf numerical schemes and sequential summation of right-hand side contributions.
- Unified addressing on tightly coupled CPU–GPU and APU platforms increases total problem size with negligible performance hit.
- FP32 compute and FP16 storage further reduce memory use while remaining numerically stable, enabled by the algorithm's well-conditioned numerics.
- Reduce memory footprint 25-fold over state-of-the-art. Improve time and energy-to-solution factors of 4 and 5.4, compared to an optimized implementation of state-of-the-art methods.
- First CFD simulation exceeding 200T grid points and 1 quadrillion degrees of freedom, improving on previous largest simulations by a factor of 20.

How big is a quadrillion???

$$1 \text{ quadrillion} = 1,000,000,000,000,000 = 1 \times 10^{15}$$

$$1 \text{ quadrillion seconds } \approx 31.7 \; \textit{million} \text{ years}$$

- Model and numerical method
- Basic implementation details
- System-specific design
- Performance and results

Model and numerical method

Navier-Stokes equations for a single fluid

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \boldsymbol{u}) = 0,$$

$$\frac{\partial (\rho \boldsymbol{u})}{\partial t} + \nabla \cdot (\rho \boldsymbol{u} \otimes \boldsymbol{u} + p\mathbf{I} - \mathbf{T}) = \mathbf{0},$$

$$\frac{\partial E}{\partial t} + \nabla \cdot [(E + p)\boldsymbol{u} - \boldsymbol{u} \cdot \mathbf{T}] = 0,$$

Old solution method:

- High-order finite volume solver
- HLLC Riemann solver
- WENO spatial reconstructions
- Requires converting between conservative and primitive variables for stability
- Expensive, nonlinear, and ill-conditioned at lower floating-point precisions

Model and numerical method

Navier-Stokes equations for a single fluid

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \boldsymbol{u}) = 0,$$

$$\frac{\partial (\rho \boldsymbol{u})}{\partial t} + \nabla \cdot (\rho \boldsymbol{u} \otimes \boldsymbol{u} + p\mathbf{I} - \mathbf{T}) = \mathbf{0},$$

$$\frac{\partial E}{\partial t} + \nabla \cdot [(E + p)\boldsymbol{u} - \boldsymbol{u} \cdot \mathbf{T}] = 0,$$

New solution method (IGR):

- Add some terms to the equation
- Solve using linear numerics and in purely conservative variables

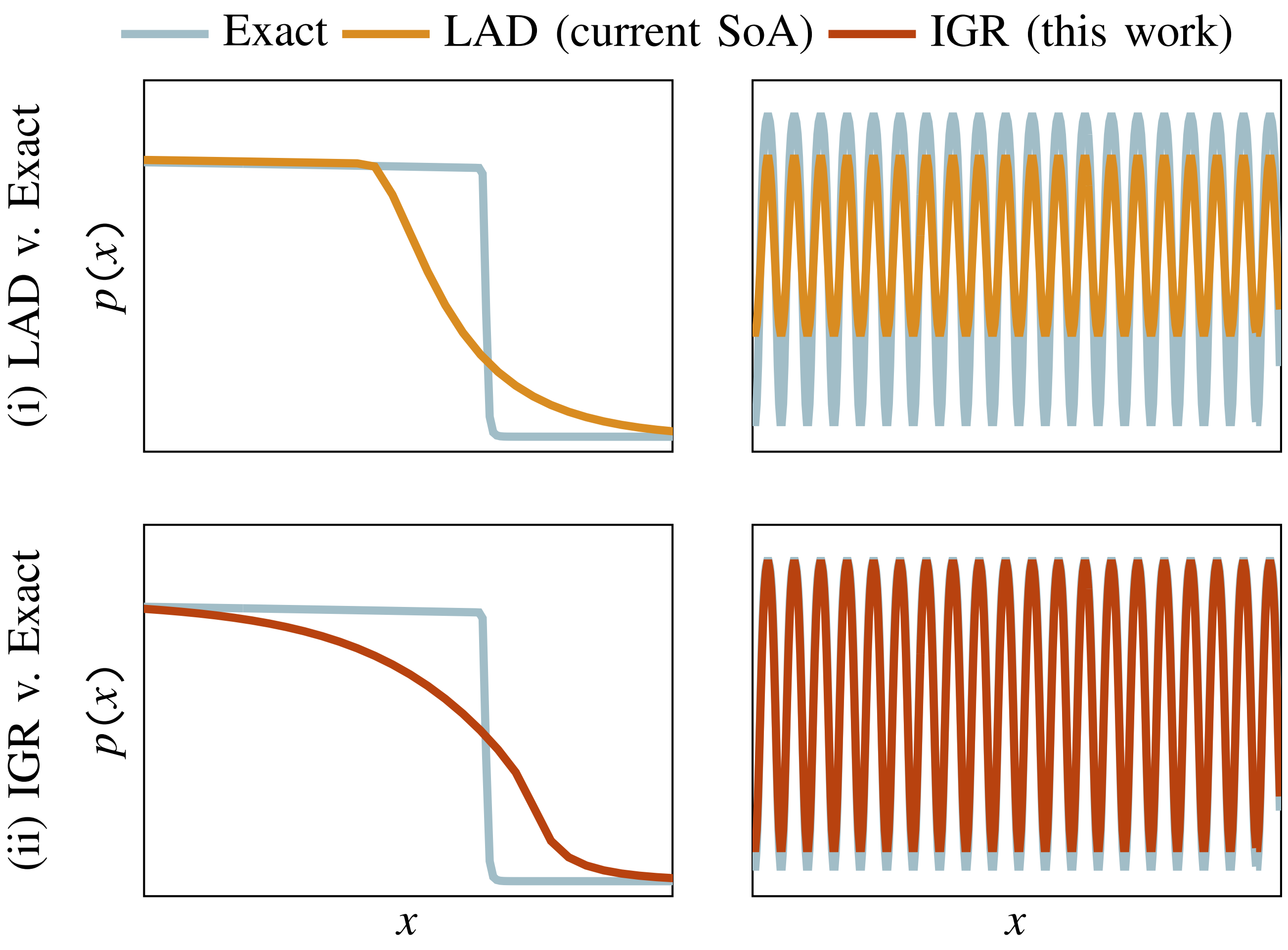
$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \boldsymbol{u}) = 0,$$

$$\frac{\partial (\rho \boldsymbol{u})}{\partial t} + \nabla \cdot (\rho \boldsymbol{u} \otimes \boldsymbol{u} + (p + \Sigma)\mathbf{I} - \mathbf{T}) = \mathbf{0},$$

$$\frac{\partial E}{\partial t} + \nabla \cdot [(E + p + \Sigma)\boldsymbol{u} - \boldsymbol{u} \cdot \mathbf{T}] = 0,$$

$$\alpha \left[\operatorname{tr}(\nabla \boldsymbol{u})^2 + \operatorname{tr}^2(\nabla \boldsymbol{u})\right] = \frac{\Sigma}{\rho} - \alpha \nabla \cdot \left(\frac{\nabla \Sigma}{\rho}\right)$$

Benefits of IGR



(a) Shock problem (b) Oscillatory problem

- Existing approach (LAD) yields solutions that are not smooth to higher orders
  - Can lead to spurious oscillations at discontinuities and dissipation of oscillatory features
- IGR replaces shocks with high-order smooth profiles to reduce oscillations near shocks and dissipation of oscillatory features

Credit: Cao and Schafer, 2024

Solution overview

3rd order SSP TVD Runge-Kutta

$$\boldsymbol{q}^{(2)} = \boldsymbol{q}^{(1)},$$

$$\boldsymbol{q}^{(1)} = \boldsymbol{q}^{(1)} + \Delta t \frac{\partial \boldsymbol{q}^{(1)}}{\partial t},$$

$$\boldsymbol{q}^{(1)} = \frac{3}{4}\boldsymbol{q}^{(2)} + \frac{1}{4}\boldsymbol{q}^{(1)} + \frac{1}{4}\Delta t \frac{\partial \boldsymbol{q}^{(1)}}{\partial t},$$

$$\boldsymbol{q}^{(1)} = \frac{1}{3}\boldsymbol{q}^{(2)} + \frac{2}{3}\boldsymbol{q}^{(1)} + \frac{2}{3}\Delta t \frac{\partial \boldsymbol{q}^{(1)}}{\partial t}$$

RHS Calculation

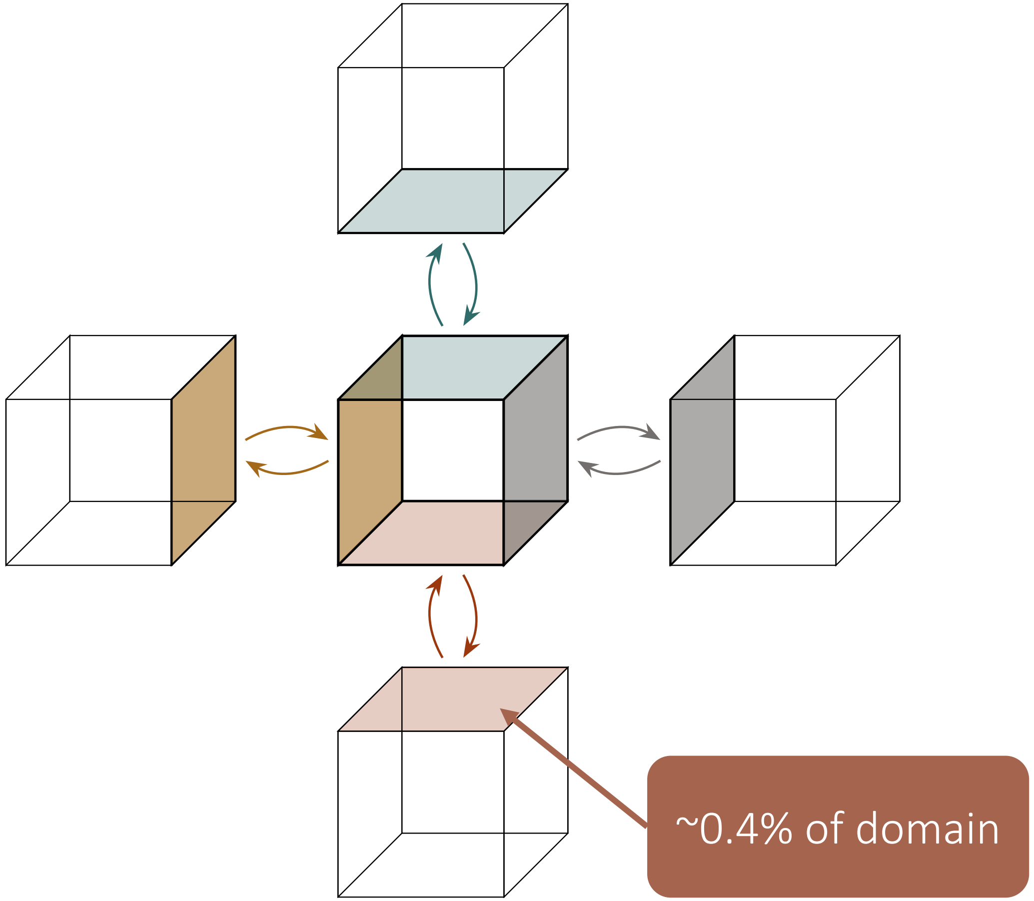
$$\frac{\partial \boldsymbol{q}}{\partial t} = \begin{cases} \frac{\partial \rho}{\partial t} = -\nabla \cdot (\rho \boldsymbol{u}), \\ \frac{\partial (\rho \boldsymbol{u})}{\partial t} = -\nabla \cdot (\rho \boldsymbol{u} \otimes \boldsymbol{u} + (p + \Sigma)\mathbf{I} - \mathbf{T}, \\ \frac{\partial E}{\partial t} = -\nabla \cdot \left[(E + p + \Sigma)\boldsymbol{u} - \boldsymbol{u} \cdot \mathbf{T} \right], \end{cases}$$

Elliptic solve for entropic pressure

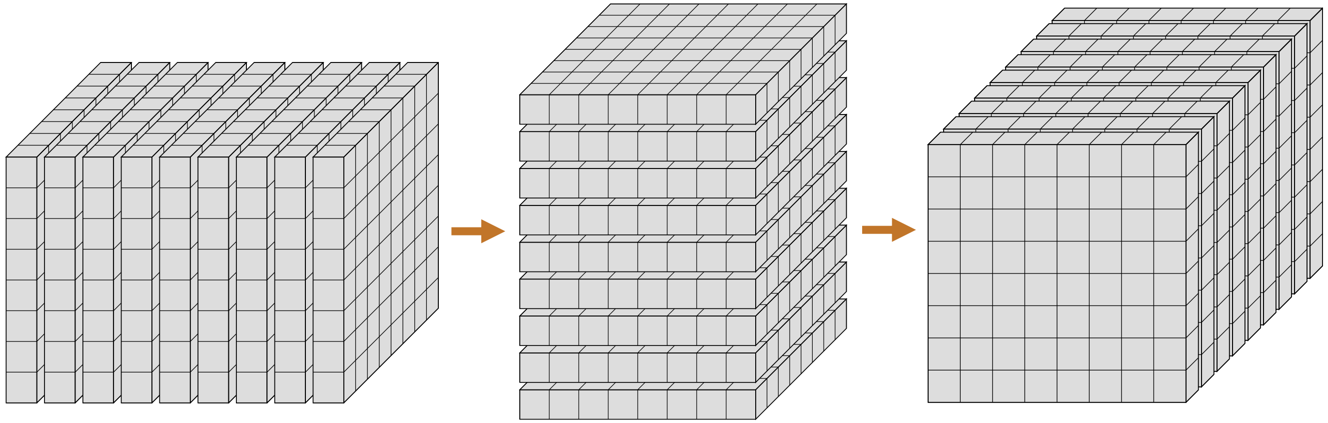
$$\frac{\Sigma}{\rho} - \alpha \nabla \cdot \left(\frac{\nabla \Sigma}{\rho} \right) = \underbrace{\alpha \left[\mathrm{tr}\,(\nabla \boldsymbol{u})^2 + \mathrm{tr}^2\,(\nabla \boldsymbol{u}) \right]}_{\Sigma_{\mathrm{rhs}}}$$

- Model and numerical method
- Basic implementation details
- System-specific design
- Performance and results

Domain decomposition and communication



Block decomposition communication pattern



Example max problem size decompositions

N_{proc}	Proc Disc.	Total (T)
128	$4 \times 4 \times 8$	0.340
384	$8 \times 8 \times 6$	1.02
1024	$16 \times 16 \times 8$	2.73
3072	$16 \times 16 \times 12$	8.18
8192	$32 \times 32 \times 16$	21.8
75264	$48 \times 49 \times 32$	200

Storage and I/O

```
restart_data
├── P1
│   ├── lusture_0.dat
│   ├── lusture_100.dat
│   └── ⋮
├── P2
│   ├── lusture_0.dat
│   ├── lusture_100.dat
│   └── ⋮
└── ⋮
```

Pros:

- Simple
- Fast

Cons:

- O(10^6) files for leadership scale simulation
- Lots of concurrent metadata creation

$$(1 \times 10^{15} \text{ scalars}) \times \left(16 \ \frac{\text{bits}}{\text{scalar}}\right) / \left(8 \times 10^{12} \ \frac{\text{bits}}{\text{terabyte}}\right) = 2000 \text{ terabytes} = 2 \ \boldsymbol{Petabytes}$$

GPU programming landscape

	OpenMP			OpenACC		
Compiler	NV GPUs	AMD GPUs	FP16 Atomics	NV GPUs	AMD GPUs	FP16 Atomics
AMD	❌	✅	✅	❌	❌	❌
CCE	✅	✅	❌	✅	✅	❌
NVHPC	✅	❌	?	✅	❌	✅*

- Modern supercomputers are procured from two hardware vendors, so portability is important!
- OpenMP is generally better supported, but OpenACC is generally faster when the necessary features are available
- If we want the best performance we can get on all hardware we need to support everything

Portable GPU offload using directive based programming and macros

Source code with macros

```
#:call GPU_PARALLEL_LOOP(collapse=3, private='[...]')
do l = 0, p            ! Z-direction
    do k = 0, n        ! Y-direction
        do j = 0, m ! X-direction
            !!> Core kernel,
            !!> O(1000) arithmetic operations
        end do
    end do
end do
#:endcall GPU_PARALLEL_LOOP
```

Preprocessor

OpenACC Code

```
!$acc parallel loop vector gange collapse(3) &
!$acc default(present) private(...)
do l = 0, Nz            ! Z-direction
    do k = 0, Ny        ! Y-direction
        do j = 0, Nx ! X-direction
            !!> Core kernel,
            !!> O(1000) arithmetic operations
        end do
    end do
end do
!$acc end parallel loop
```

OpenMP Code

```
!$omp target teams distribute parallel do simmd &
!$omp defaultmap(firstprivate:scalar) &
!$omp defaultmap(tofrom:aggregate) &
!$omp defaultmap(present:allocatable) &
!$omp defaultmap(present:pointer) &
!$omp collapse(3) private(...)
do l = 0, Nz            ! Z-direction
    do k = 0, Ny        ! Y-direction
        do j = 0, Nx ! X-direction
            !!> Core kernel,
            !!> O(1000) arithmetic operations
        end do
    end do
end do
!$omp end target teams distribute parallel do
```

- OpenMP and OpenACC are generated from one version of source code
- Developers don't need in depth understanding of directive tools

- Model and numerical method
- Basic implementation details
- System-specific design
- Performance and results

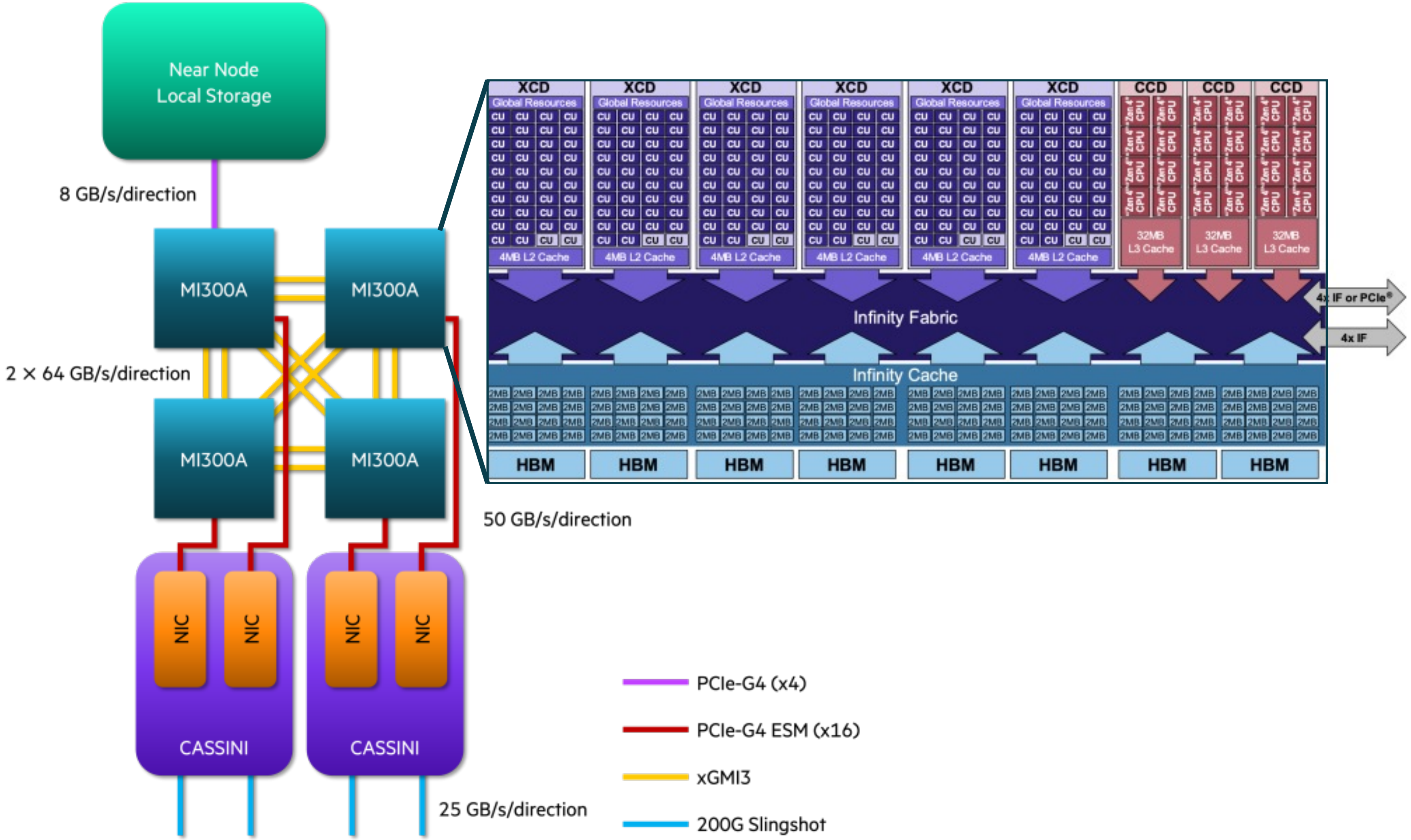
System Summary

	Node Configuration	# Nodes	Memory [Node, System]	Peak Power	Rmax	TOP500
LLNL El Capitan	4 AMD MI300A APU	11136	[512 GB, 5.6 PB] APU	34.8 MW	1742 PFLOPs	1
OLCF Frontier	4 AMD MI250X GPU 1 AMD Trento CPU	9472	[512 GB, 4.8 PB] GPU [512 GB, 4.8 PB] CPU	24.6 MW	1353 PFLOPs	2
CSCS Alps	4 NVIDIA GH200 (4 Grace CPU, 4 Hopper GPU)	2688	[384 GB, 1.0 PB] GPU [480 GB, 1.3 PB] CPU	7.1 MW	435 PFLOPs	8

- All systems of interest are within the top 10 fastest computers measured by the HPL benchmark
- CSCS Alps is “little brother” to 6k node JSC Jupiter which is ranked 3rd now
- >95% of total system memory is used on El Capitan and Frontier
- >85% of total system memory is used on Alps

- Shared critical features:
  - Ability to leverage unified address space between CPU and GPU
  - High performance networking and leadership class scale

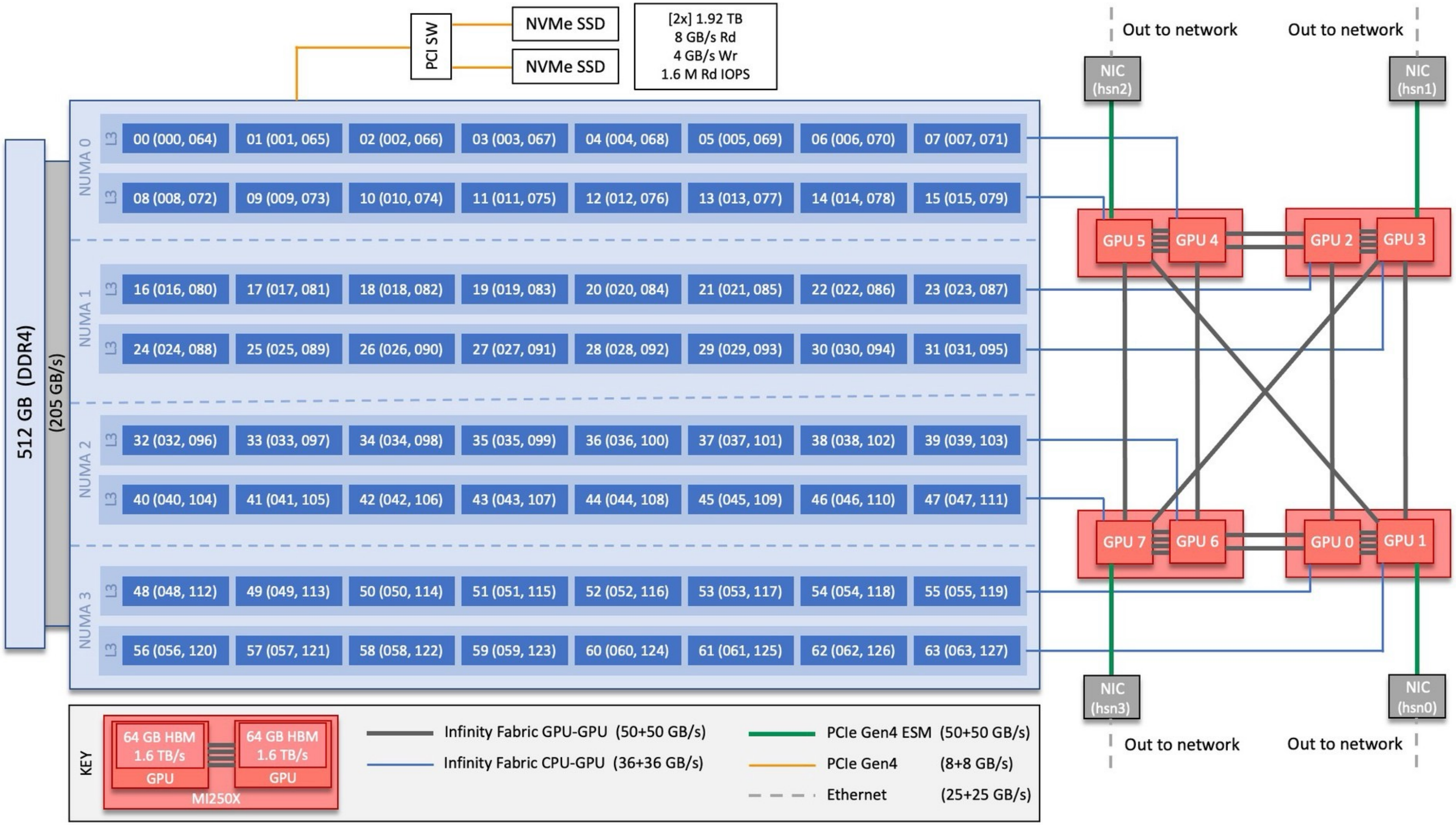
Tightly coupled GPU/APU architectures (AMD MI300A, LLNL El Capitan)



- 24 Zen4 CPU cores bonded to 6 AMD XCD chiplets
- 4 APUs per node
- Memory is universally addressable in address space and physical space
- Zero host-to-device transfers to perform because the host and device are one

Credit:https://hpc.llnl.gov/documentation/user-guides/using-el-capitan-systems/hardware-overview

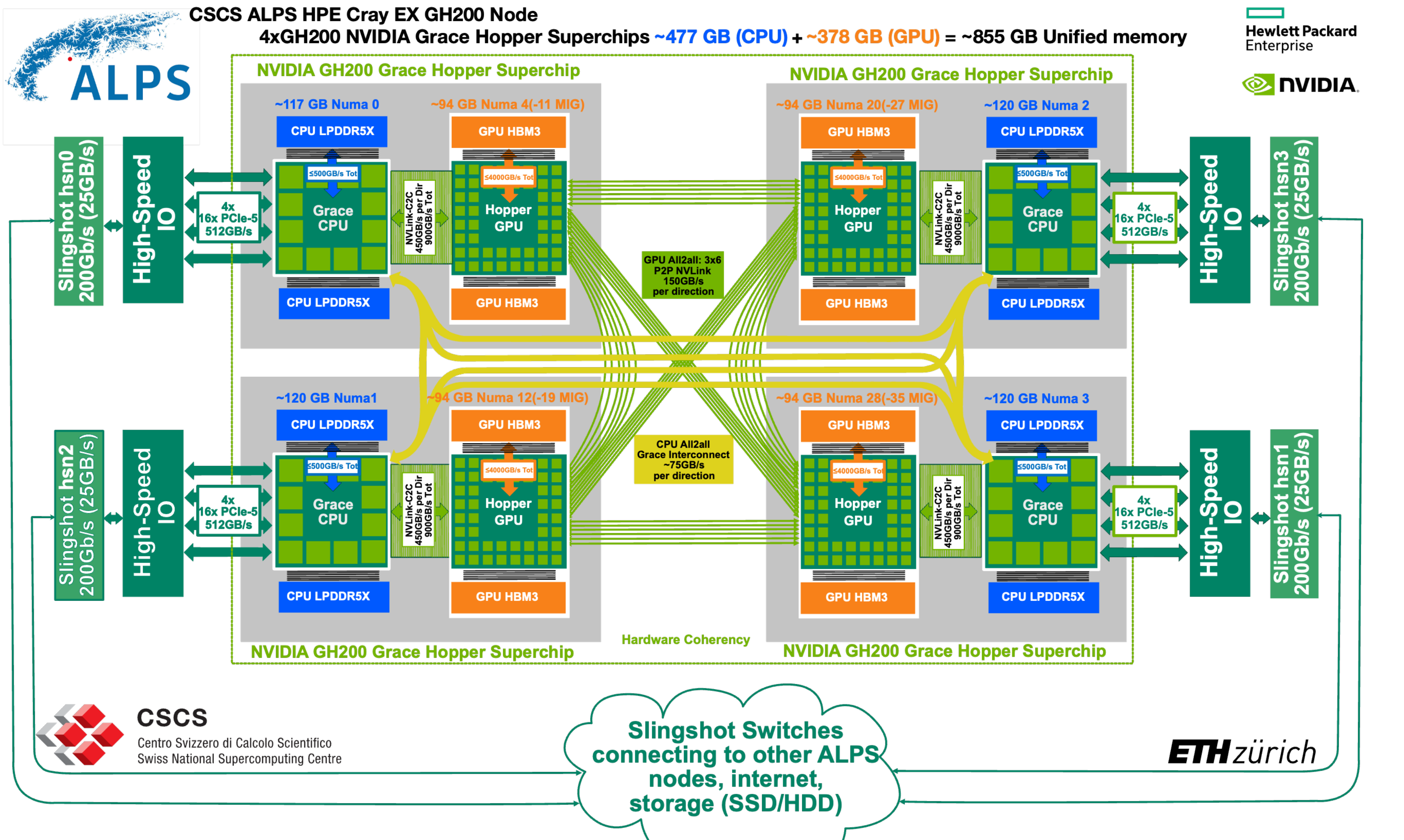
Tightly coupled GPU/APU architectures (AMD MI250X, OLCF Frontier)



- 4 Mi250X GPUs (8 GCDs) connected to one CPU via 36+36 GB/s links
- 1 CPU and 4 GPUs per node
- Each chip has its own memory
- Unified address space made possible through careful memory allocation and infinity fabric

Credit: https://docs.olcf.ornl.gov/systems/frontier_user_guide.html

Tightly coupled GPU/APU architectures (NVIDIA GH200, CSCS Alps)



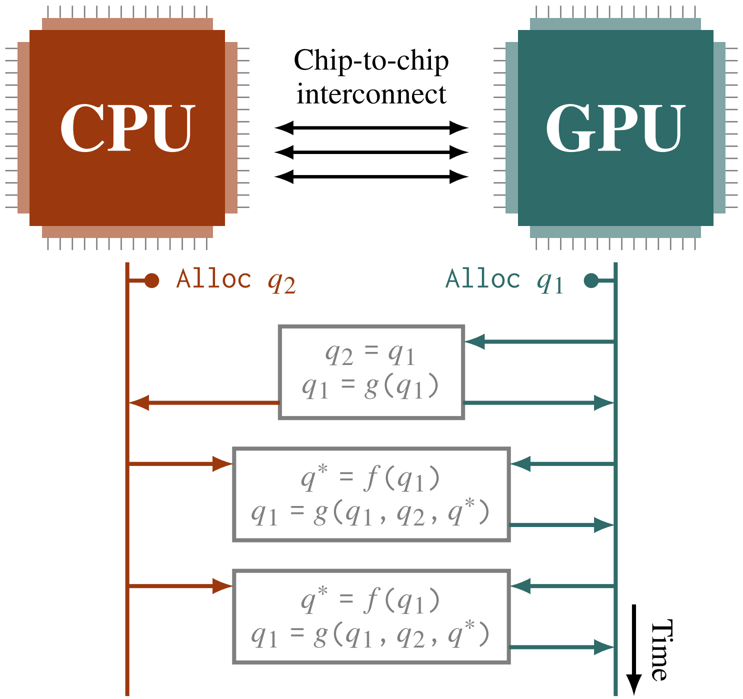
- Grace CPU and Hopper GPU connected via 900 GB/s chip-to-chip (C2C) link
- 4 CPUs and 4 GPUs per node
- Each chip has its own memory
- Unified address space made possible via high-speed interconnect, though memory regions are physically separate

Credit: https://docs.cscs.ch/alps/hardware/#nvidia-gh200-gpu-nodes

Unified memory strategy

MI250x - Allocate q_2 on the using **`hipMallocManaged`** and advise runtime *not* to make device copy

GH200 – Allocate q_2 and use **`cudaMemAdvise`** to keep memory on the host



MI250x - Allocate q_1 on the GPU using **`hipMalloc`** and advise runtime *not* to make a host copy

GH200 – Allocate q_1 and pin memory with **`cudaMemAdvise`**

- GH200's 900 GB/s link allows for storage of Σ and Σ_{rhs} on the host as well, further increasing problem size

- Model and numerical method
- Basic implementation details
- System-specific design
- Performance and results

Performance: Grindtime

Grindtime

Device	Baseline (in-core)	IGR (in-core)	IGR (unified)	
GH200	16.89	3.83	4.18	FP64
MI250X GCD	69.72	13.01	19.81	
MI300A	29.50	†—	7.21	
GH200	*N/A	2.70	2.81	FP32
MI250X GCD	*N/A	9.12	13.03	
MI300A	*N/A	†—	4.19	
GH200	*N/A	3.06	3.07	FP16/32
MI250X GCD	*N/A	22.63	24.71	
MI300A	*N/A	†—	17.39	

*Numerically unstable; †MI300A is always unified

Normalized by nanoseconds/grid cell/time step

- Baseline numerics unstable in lower precision
- AMD mixed precision slow due to using AMD's beta compiler and a pre-release of NVHPC SKD 25.9

- ~9% overhead in double precision on NVIDIA due to compiler regression
- <5% slowdown in single and mixed precision on NVIDIA thanks to 900 GB/s link
- 40-50% overhead in double and single precision on MI250x due to slower 36+36 GB/s link

- Up to 6x reduction in time to solution

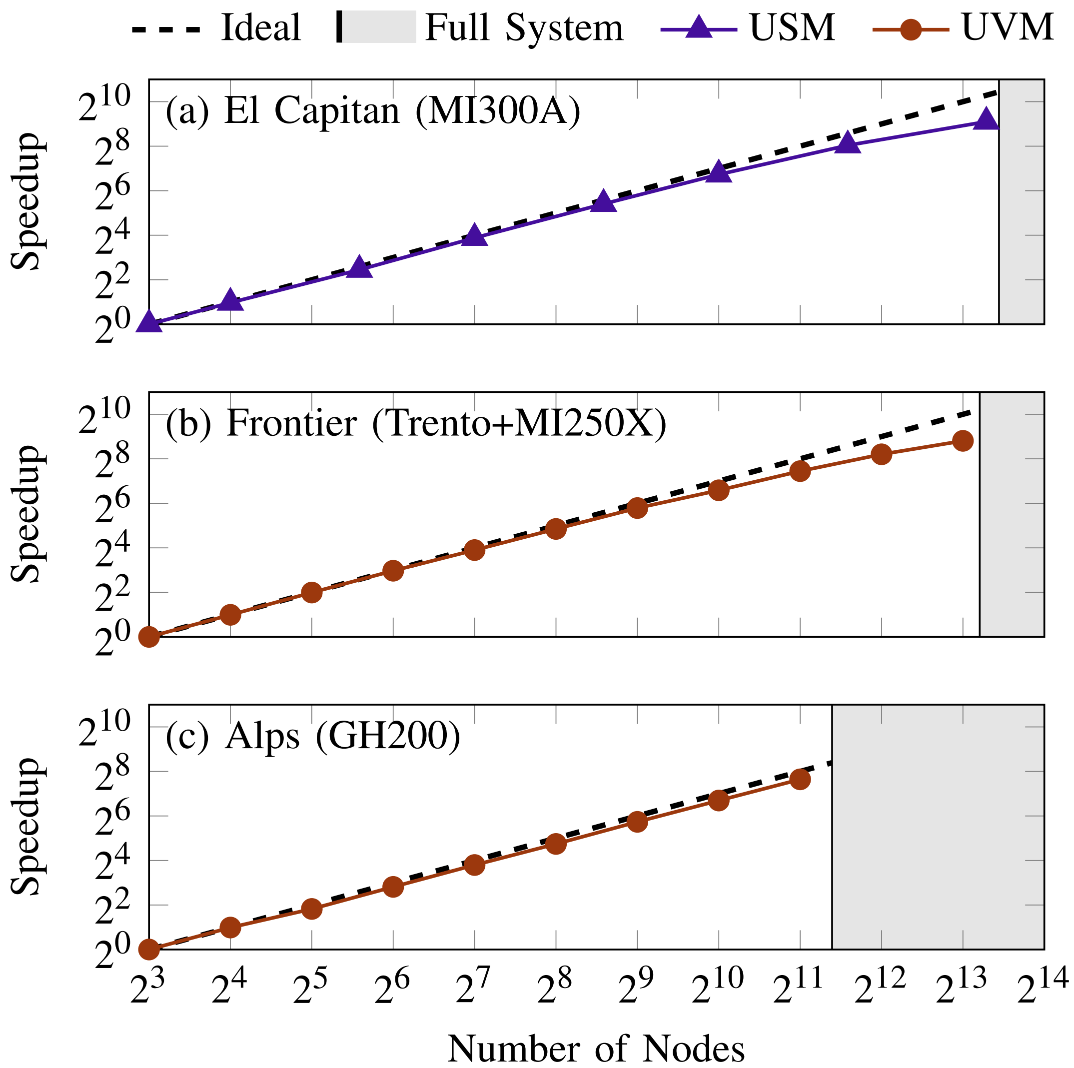
Performance: Power

Power

Energy (μJ)	El Capitan (MI300A)	Frontier (MI250X)	Alps (GH200)
Baseline	15.24	10.67	9.349
IGR	3.493	1.982	2.466

- Calculated by sampling `nvidia-smi` and `rocm-smi` to get a steady state wattage and then multiply by time
- Measured in in double precision for an apples-to-apples comparison
- Findings show power consumption is proportional to runtime as a first-order approximation
- GH200 uses more power with novel numerics than current state-of-the-art, but algorithm is faster

Performance: Strong scaling



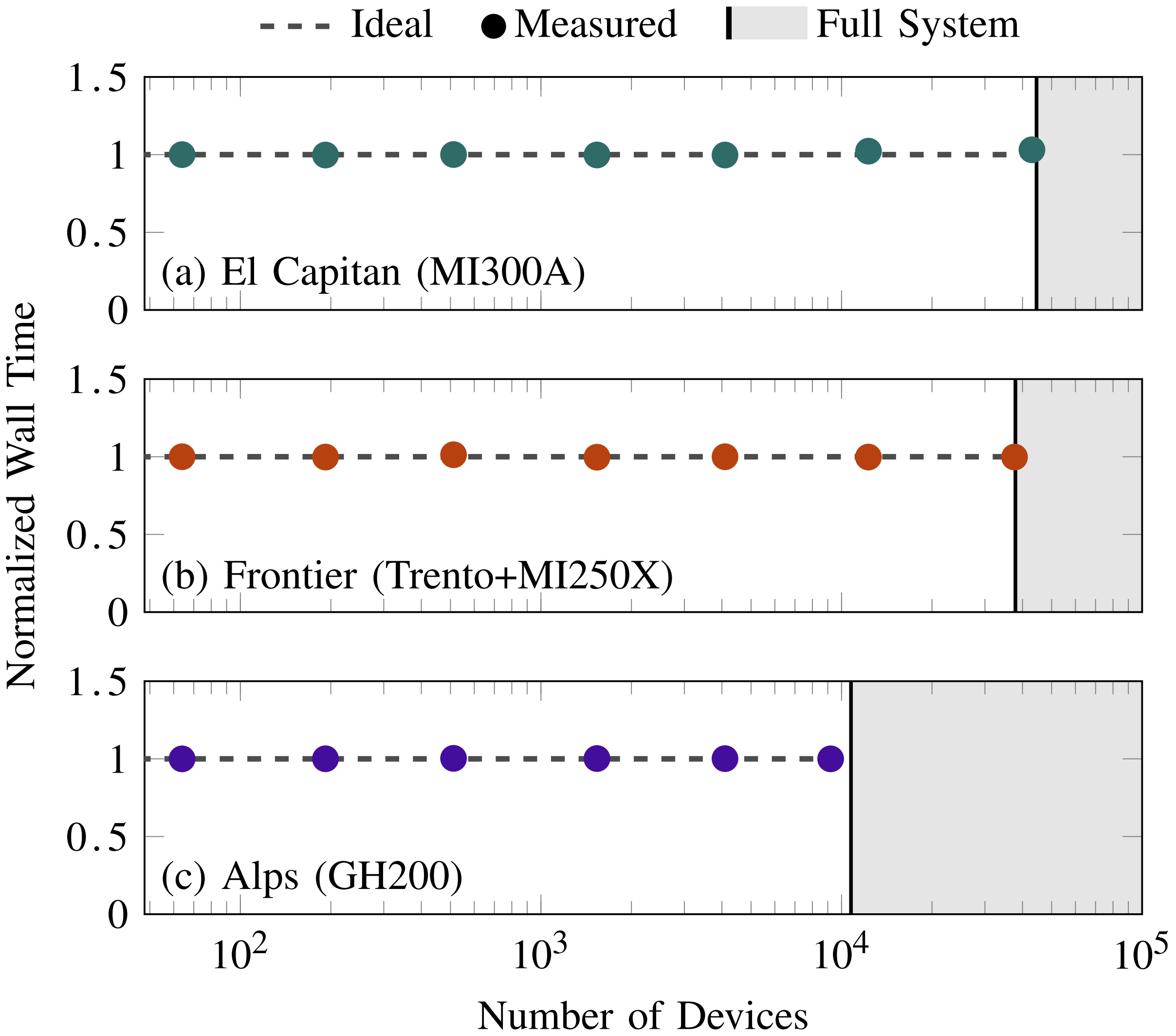
- 8-node base case does internode communication in all three physical dimensions
- Higher efficiency results in shorter time to solution for a given problem size

$$\text{Efficiency} = \frac{\text{Actual Speedup}}{\text{Ideal Speedup}}$$

Strong Scaling Efficiencies

Systemn	Base	32× Efficieny	Full System	
El Capitan	84 B	90%	10000 Nodes	44%
Frontier	170 B	90%	8192 Nodes	45%
Alps	134 B	86%	2048 Nodes	78%

Performance: Weak scaling



- 16-node base case does internode communication in all three physical dimensions (and makes for nice spacing)
- Efficiencies are all approx. 100%

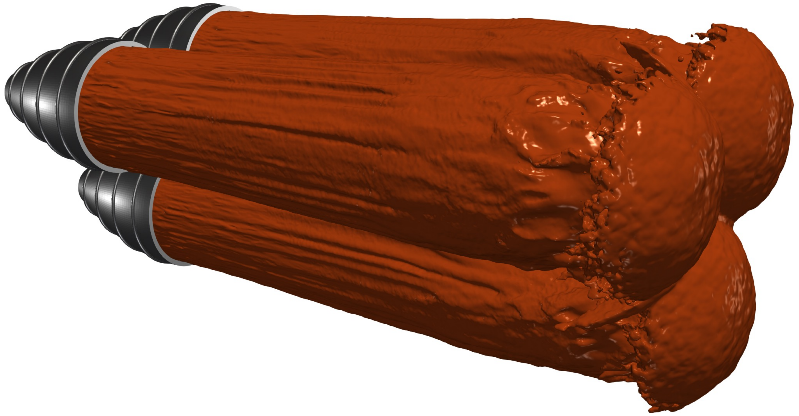
$$\text{Efficiency} = \frac{T_{\text{base case}}}{T}$$

Weak Scaling Efficiencies

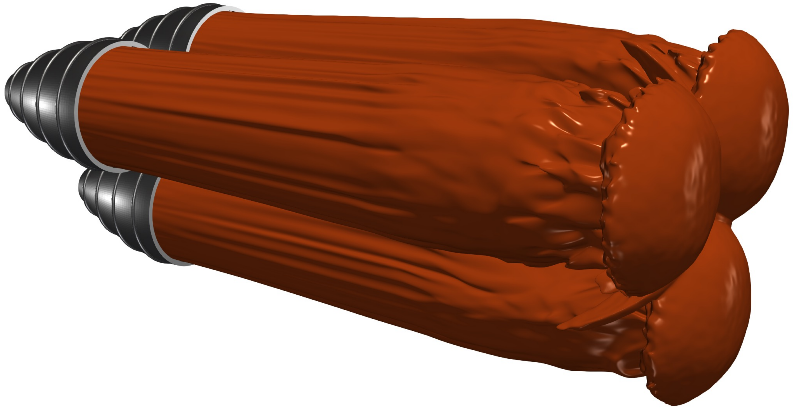
System	Base	Max	Increase	Efficiency
El Capitan	168 B	113 T	672×	97%
Frontier	341 B	200 T	588×	99%
Alps	268 B	38 T	143×	99%

Results: Precision comparison

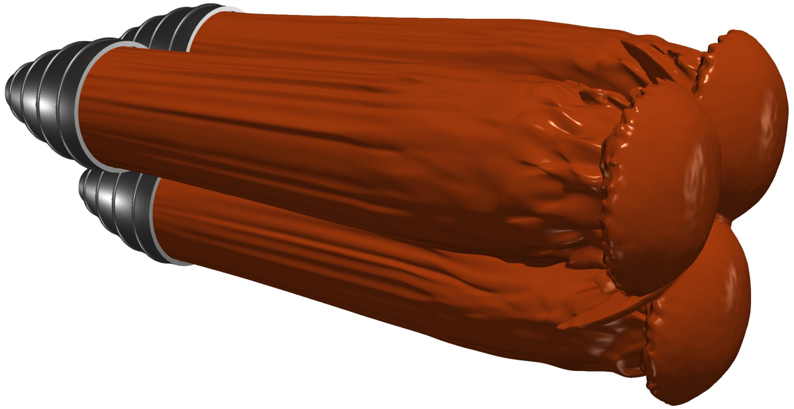
FP16/FP32



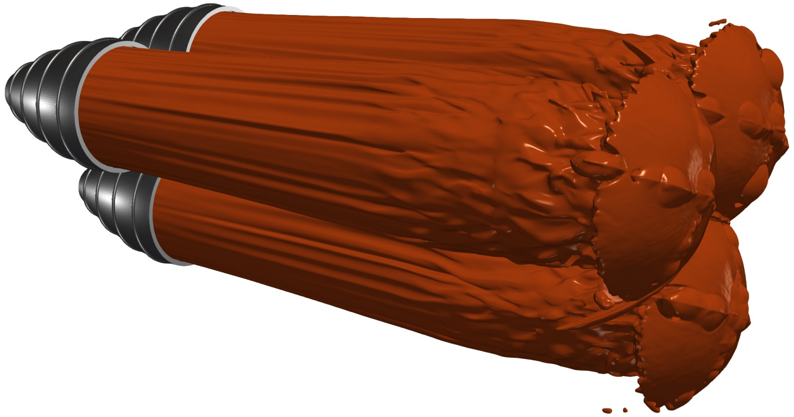
FP32



FP64



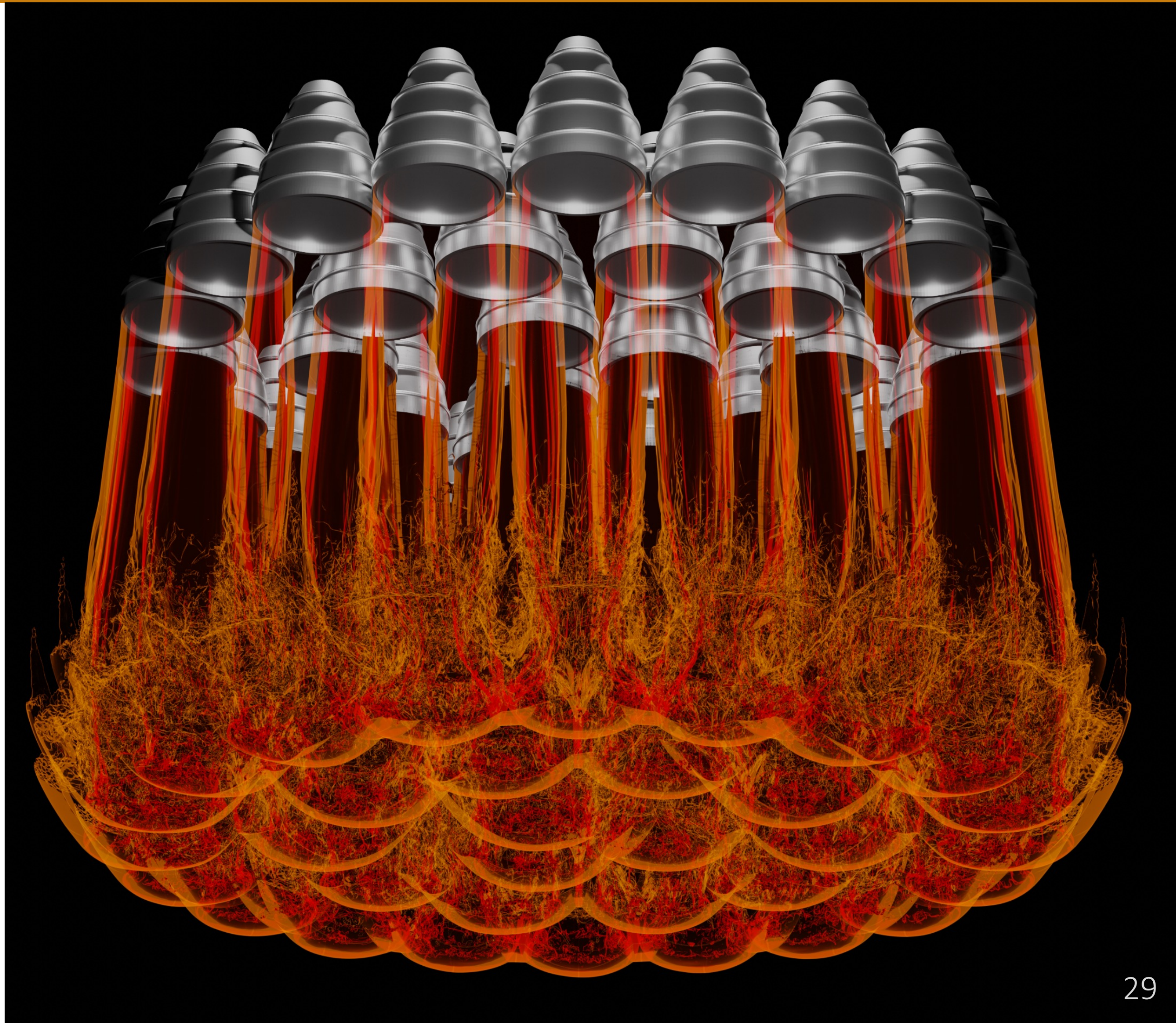
FP64 (baseline numerics)



- FP64 and FP32 visually indistinguishable
- FP16/FP32 is visually different, though the main flow features are captured
- Numerical error leads to early onset of hydrodynamic instabilities
- FP64 baseline has grid aligned artifacts due to gird aligned shock capturing

Results: Super heavy booster configuration

- 3.3 trillion grid cells
- 16.5 trillion degrees of freedom
- 16 hours on 9200 GH200 GPUs



Lessons learned

- NVIDIA compilers are great, but they let you do a lot of things that aren’t in the standards, so supporting new compilers can be challenging
- Compiler support for directive-based programming with lower floating-point precisions is still developing
- Heterogenous architectures are cool and allow you to do some interesting things
- Doing something no one has ever done before is difficult and time consuming

Code

Preprint

Questions?

Acknowledgements

Oak Ridge National Laboratory | Leadership Computing Facility

AMD

Lawrence Livermore National Laboratory

NVIDIA

Hewlett Packard Enterprise | CRAY

CSCS