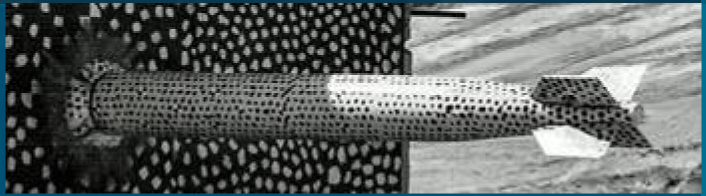
SAND2022-9512C

Sandia National Laboratories

A performance portable implementation of high-order, entropy-stable spectral collocation schemes for compressible turbulent flows



July 31^{st} to August 5^{th}, 2022

PRESENTED BY

Jerry Watkins, Travis Fisher, and Wyatt Horne

15^{th} World Congress on Computational Mechanics
8^{th} Asian Pacific Congress on Computational Mechanics
Yokohama, Japan (Virtual)

SAND

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Outline

- Introduction
  - Motivation – High-fidelity simulations
  - Motivation – Exascale computing
- High-order, matrix-free methods and performance portability
  - High-order, entropy-stable methods
  - High-order communication and operators
  - High-order methods on modern hardware
  - Matrix-free methods
  - Matrix-free methods on modern hardware
  - Performance portable C++ frameworks
- Numerical Results
  - Case/Study setup
  - Taylor-Green Vortex
  - Flat plate

Motivation

Why are we interested in performance and portability?

High-fidelity simulations

High-fidelity simulations on exascale systems for analysis/design in hypersonics

Multi-fidelity design tools

Direct Numerical Simulation (DNS)
- Purpose: Model Development and Uncertainty Quantification
- Methods: High-order structured or unstructured methods

Wall-modeled LES and hybrid RANS/LES
- Purpose: Higher-fidelity engineering analysis
- Methods: High-order or low-dissipation finite volume

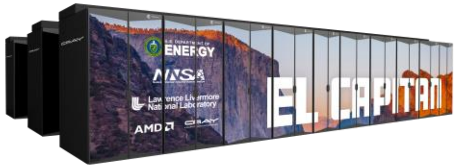
RANS
- Purpose: Engineering calculations
- Methods: Second-order finite volume

Reduced-order and semi-empirical models
- Purpose: Engineering
- Methods: Various

Target systems

LANL Trinity
Intel (KNL)

Sierra
NVIDIA (V100)

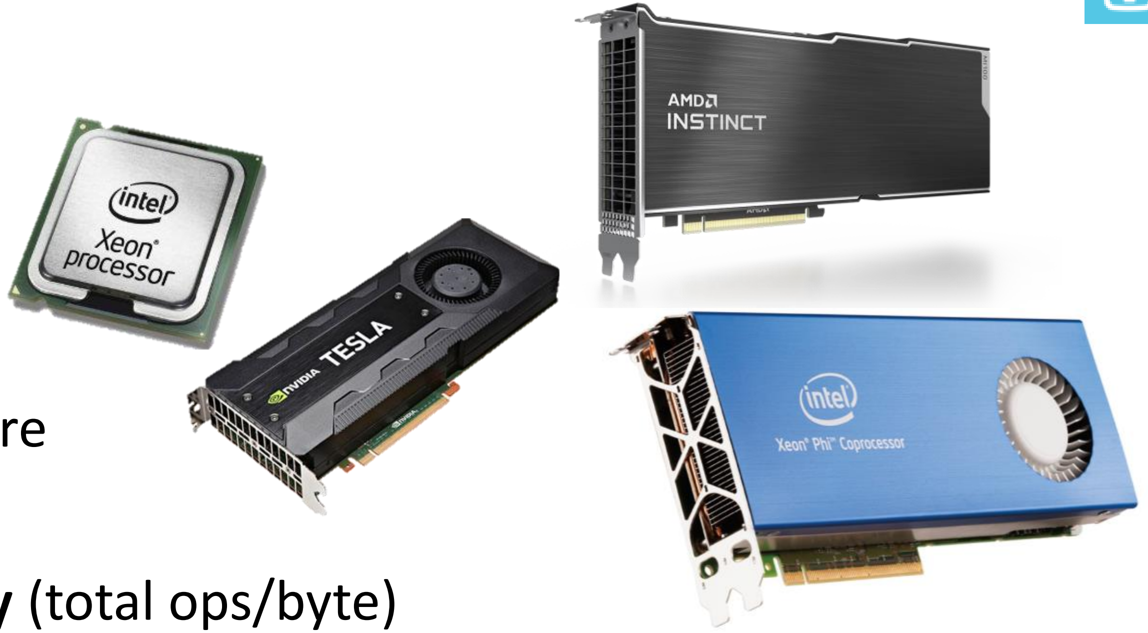


El Capitan
AMD

CROSSROADS

Crossroads
Intel

Exascale computing



<u>Challenges:</u>

- Diverse set of HPC vendors and architectures
  - Intel, AMD, NVIDIA, IBM, ARM-based
  - CPUs with vector processing; GPUs
- Software life cycle is much longer than hardware

<u>Different architectures, trend remains the same</u>

- Need algorithms with **high arithmetic intensity** (total ops/byte)
- Need fundamental **abstractions** during code development

<u>Performance portability</u>: A reasonable level of performance is achieved across a wide variety of computing architectures with the same source code.

<u>Approaches:</u>

- **Libraries** – High-level abstractions with specified input/output (e.g. BLAS)
- **Task-based** – Data-centric abstractions for mapping tasks to resources (e.g. Legion)
- **MPI+X** – Algorithmic-level abstractions for distributed (MPI) and shared (X) memory parallelism (e.g. **Directives**: OpenMP, OpenACC; **Frameworks**: Kokkos, RAJA, OCCA)

High-order, matrix-free methods and performance portability

What strategies are we using for high-order, matrix-free methods and performance portability?

High-order, entropy-stable methods

Entropy Stable Methods → Shock Capturing → Evaluate Turbulent Dissipation

Entropy Stable Summation-by-Parts Methods:

- Multi-block structured finite difference (**HOFD**)
- Unstructured spectral collocation elements (**SCE**)

Shock capturing:

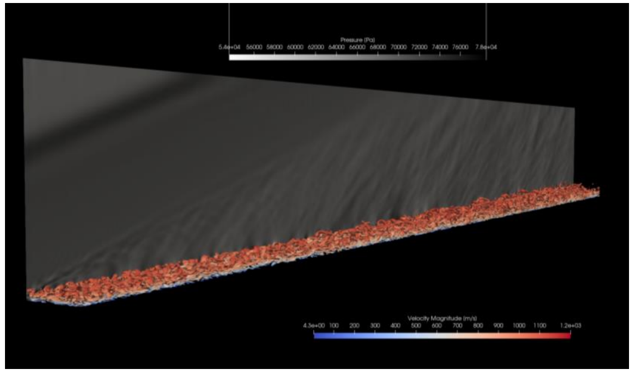
- WENO
- Artificial viscosity
- Hybridized with Larsson shock sensor

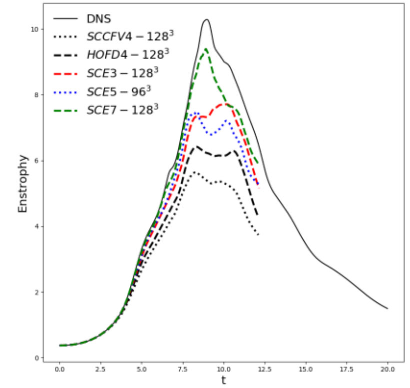
Evaluate Turbulent Dissipation:

- Need accurate and robust methods

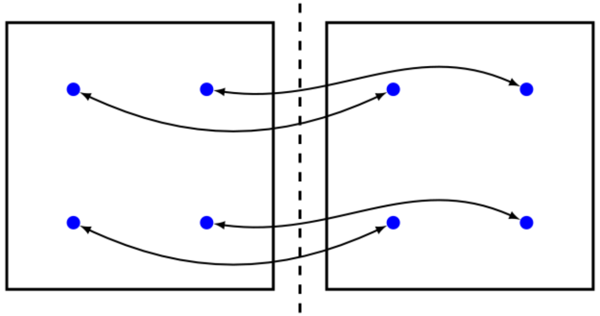
Where do discontinous Galerkin (DG) methods fit?

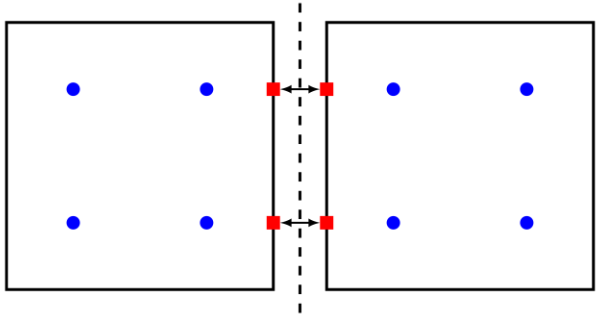
- **SCE** schemes are **nodal DG schemes** where nonlinear operators are used in place of linear operators to achieve entropy stability
- **Entropy stability** is used to help ensure **robustness** in the presence of **shocks**





High-order communication and operators





Communication:

- Ghost cell communication (LG)
- Ghost face communication (LGL)

Three major operators: Volume, Interface and Boundary

Linear Operators

$$\mathcal{D}w$$

- Gradient operators
- Matrix-vector operations in each cell (matrix-matrix including cells)
- w vector is reused for local assembly

Nonlinear Operator (Entropy Stability)

$$[\mathcal{D} \circ \mathcal{F}]\,1$$

- Flux divergence operators
- Batch vector inner product in each cell (Batch vector-matrix including cells)
- $\mathcal{F}$ matrix is not reused

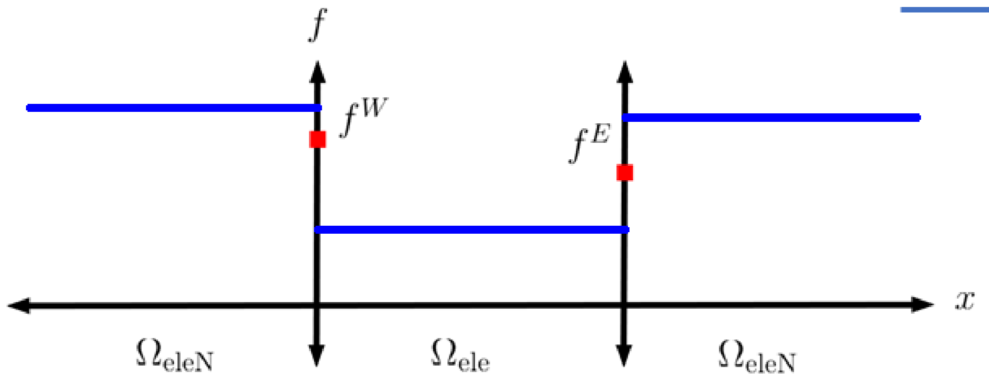
Two strategies used to avoid race conditions: **graph coloring** and **atomics**

High-order methods on modern hardware

Higher arithmetic intensity to efficiently utilize modern hardware

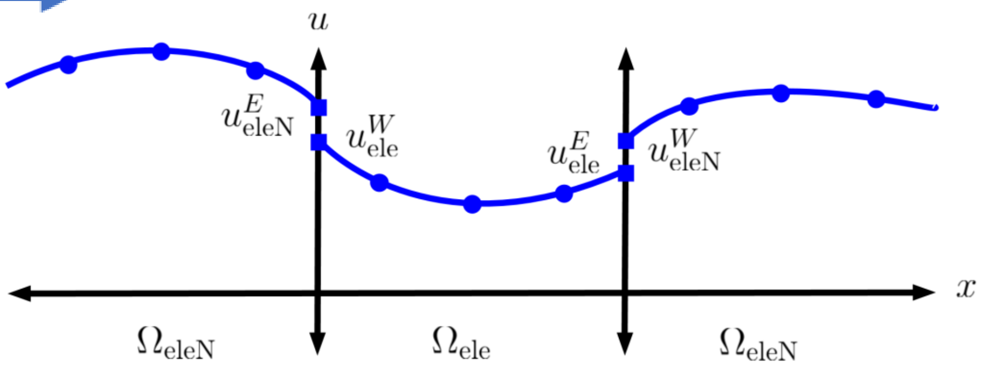
Example: No extrapolation operator in finite volume

Finite Volume



Higher FLOPS per byte →

Nodal DG (LG)



Increasing polynomial order

- More operations per degree of freedom
  - Increases computational throughput
- Majority of operations are element-local
  - Allows for efficient use of shared memory

- Improves strong scaling; reduces error
- **Challenges:**
  - Diminishing returns
  - Better performance given a fixed error metric

Matrix-free methods

Time Stepping Schemes:

Semi-discrete equation

Solution

Divergence of the flux

$$\frac{\partial \boldsymbol{U}}{\partial t} = -\boldsymbol{V}^{-1}\, \hat{\nabla}^{\delta} \cdot \hat{\boldsymbol{F}} \qquad R(\boldsymbol{U}) = -\boldsymbol{V}^{-1}\, \hat{\nabla}^{\delta} \cdot \hat{\boldsymbol{F}}$$

Determinant geometric Jacobian matrix

Residual

Runge-Kutta methods

$$\boldsymbol{U}^{n+1} = \boldsymbol{U}^{n} + \Delta t^{n} \sum_{s=1}^{N_{\text{stages}}} b_s \boldsymbol{R}_s$$

Stages can be solved sequentially (No linear system)

BDF1 for steady-state

$$\Delta \boldsymbol{U}^{m} = \Delta \mathbf{T}^{m}\, R\left(\boldsymbol{U}^{m+1}\right)$$

$$\left(\left(\Delta \mathbf{T}^{m}\right)^{-1} - \frac{\partial \boldsymbol{R}^{m}}{\partial \boldsymbol{U}^{m}} \right) \Delta \boldsymbol{U}^{m} = \boldsymbol{R}^{m}$$

Residual Jacobian matrix (Large sparse matrix)

Explicit methods (Ex: RK44)

- Pros:
  - No matrix and linear system solve required
  - Performance limited by residual
- Cons:
  - Time step often limited by numerical stability
  - Difficult to determine reliable time step
  - High-order time step restriction h/P^2

Implicit methods (Ex: BDF1)

- Pros:
  - Time step tuned to accuracy
  - More computation per sequential time step
- Cons:
  - Matrix and linear system solve required
  - Nonlinear stability is not guaranteed
  - High-order matrix size P^3

Matrix-free methods on modern hardware

Jacobian-free Newton-Krylov:

- Using GMRES to solve the linear system only requires matrix-vector products
  - Less memory (no need to store a large matrix) which allows an increase in utilization
  - Less data movement (no need to assemble a large matrix) which allows efficient bandwidth use
  - More computation (evaluate matrix-vector product at each linear iteration)
- May need matrix if preconditioning is required

Matrix-free approximate

Use Frechet derivative

- Pros:
  - Performance limited by residual
- Cons:
  - Residual evaluation at each linear iteration
  - Approximation may limit stability

Matrix-free exact

Use automatic differentiation (AD)

- Pros:
  - Quadratic convergence at best
- Cons:
  - AD evaluation at each linear iteration
  - Nonlinear stability is not guaranteed
  - More difficult to solve for stiff equations

Performance portable C++ frameworks

<u>MPI+X:</u> C++ frameworks within **Trilinos** for performance portability

- Automatic differentiation (***Sacado***)
- Distributed memory linear algebra (***Tpetra***)
- Shared memory parallelism (***Kokkos***)

Abstract **data layouts** and **hardware features** on current and future architectures

- Allocation: `U = Kokkos::DualView<double***[5]>(Ncells,Nspts,Nv)`
- Memory transfer: `U.modify_host(); U.sync_device();`
- Memory layout: `Kokkos::LayoutLeft` (col-major)
- Data parallelism: `Kokkos::parallel_for(policy, functor)`
  - `policy` defines iteration range: `Kokkos::RangePolicy(N)`
  - `functor` defines function to be parallelized
- SIMD performance portability: `SIMD_Double`

Allows researchers to focus more on **algorithm development** instead of **architecture specific programming**

https://github.com/trilinos/Trilinos/

https://github.com/kokkos/kokkos/

Numerical Results

How well do performance-portable, high-order, matrix-free methods perform?

Case setup

<u>Taylor-Green Vortex (TGV):</u>

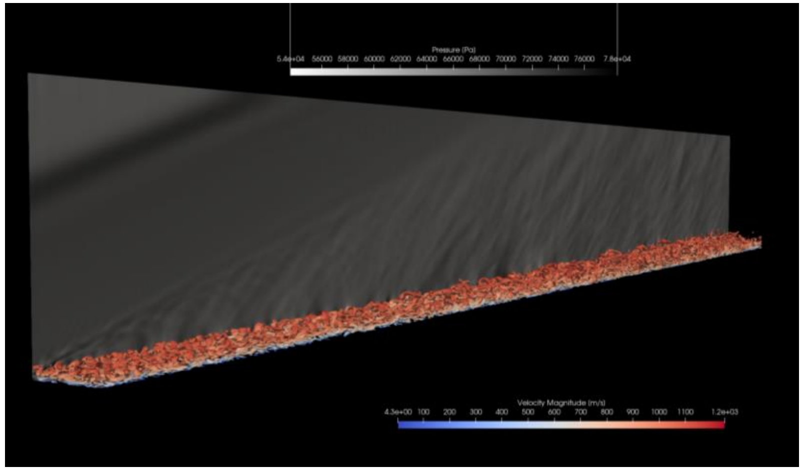
- 30s simulation time, explicit time step control

<u>Mach 0.2 Turbulent Flat Plate Boundary Layer:</u>

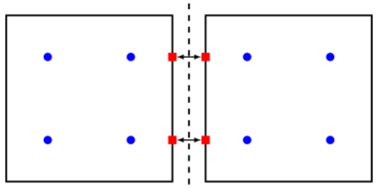
- Steady-state simulation, finite volume, matrix-free

<u>Mach 3.5 Flat Plate Boundary Layer ILES:</u>

- Synthetic turbulent inflow
- Shock capturing: Shock sensor limiting artificial viscosity
- BDF2 implicit time integration
- Low-order preconditioned Jacobian-free Newton-Krylov



Study setup



Methods:

- Structured cell-centered finite volume (**SCCFV**)
- Structured high-order finite difference (**HOFD**)
- Unstructured spectral collocation element (**SCE**)
  - P = 1-7; LGL

MPI+X Notation

$$r(\text{MPI} + j\text{X}), \quad \text{X} \in \{\text{OMP}, \text{OMPV}, \text{GPU}\}$$

r = # MPI ranks
j = # OpenMP threads or GPUs/rank
X = architecture for shared memory parallelism

Node Architectures

- Intel Haswell (**HSW**) – 32 cores, 64 threads, AVX2 (4 doubles)
- Intel Knights Landing (**KNL**) – 64 cores, 256 threads, AVX512 (8 doubles)
- Intel Cascade Lake (**CLX**) – 48 cores, 96 threads, AVX512 (8 doubles)
- ARM64 Cavium ThunderX2 (**TX2**) – 56 cores, 112 threads, Neon (2 doubles)
- NVIDIA Volta (**V100**) – 4 GPUs, Cuda (no simd)

Taylor-Green Vortex (TGV)

Setup:

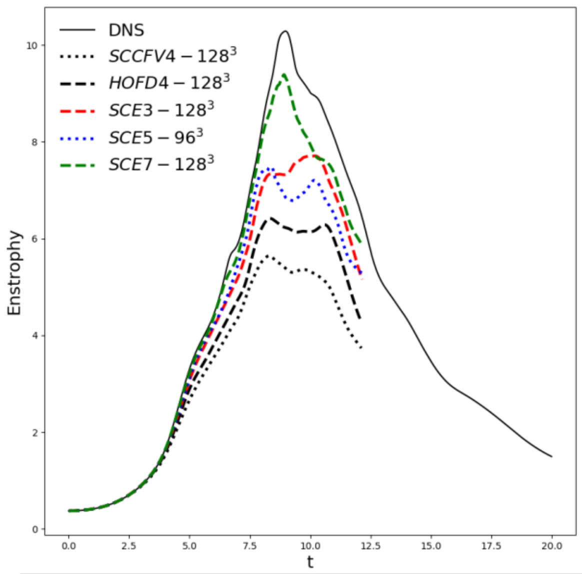
$$Figure\ of\ Merit = \frac{T_{SCCFV} \cdot err^{\varepsilon}_{SCCFV}}{T \cdot err^{\varepsilon}}$$

- T - Wall-clock time for 30s simulation time (s)
- $err^{\varepsilon} = \int |\varepsilon(t) - \varepsilon_{ref}(t)|^2 dt$ - Enstrophy error
- Reference: Spectral element solution, 512^3
- Note: results are architecture independent

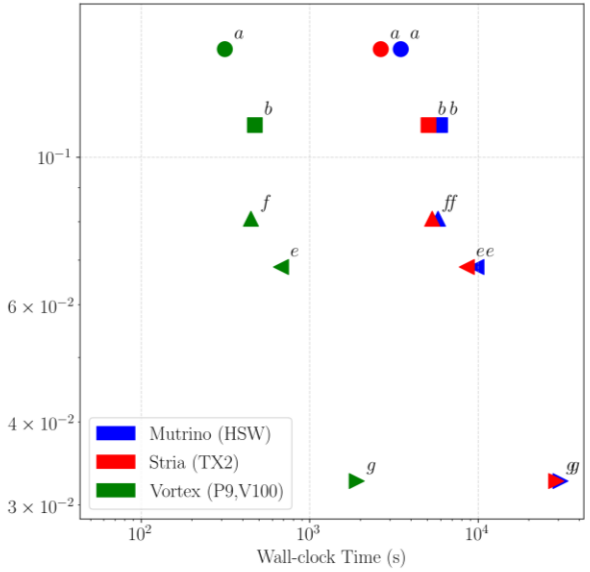
Results:

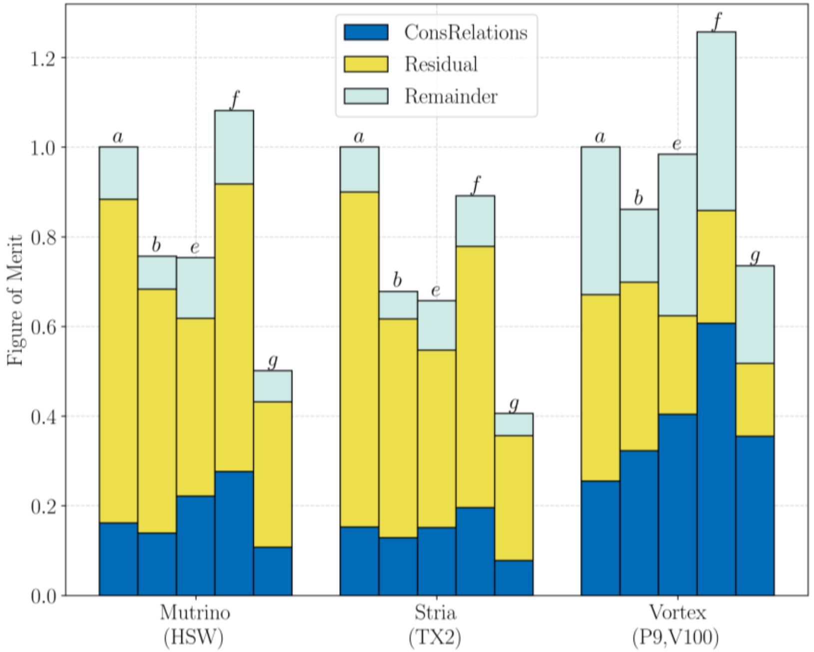
- Current optimal is near **SCE5**
- High-order performing better on GPUs
- Bottlenecks
  - CPU – Residual (computation)
  - GPU – ConsRelations (gradient, communication)
  - Remainder also needs more profiling/improvement

Enstrophy vs. t



Enstrophy Error vs. Wall-clock time





Discretization - DoF
- a: SCCFV4 - 128^3
- b: HOFD4 - 128^3
- e: SCE3 - 128^3
- f: SCE5 - 96^3
- g: SCE7 - 128^3

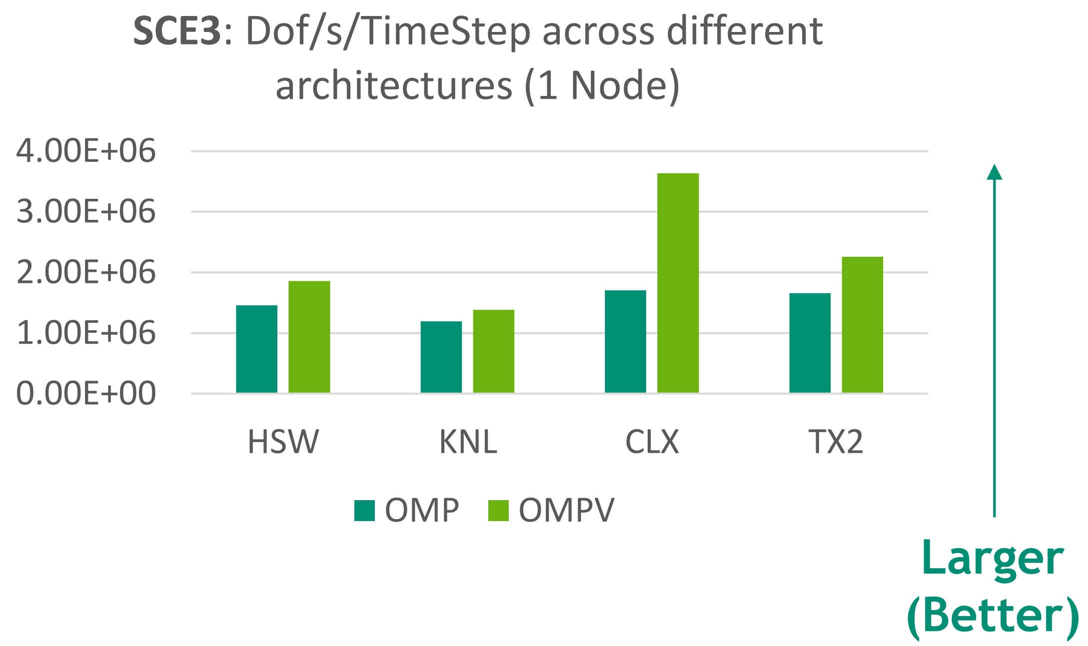
SIMD performance on TGV

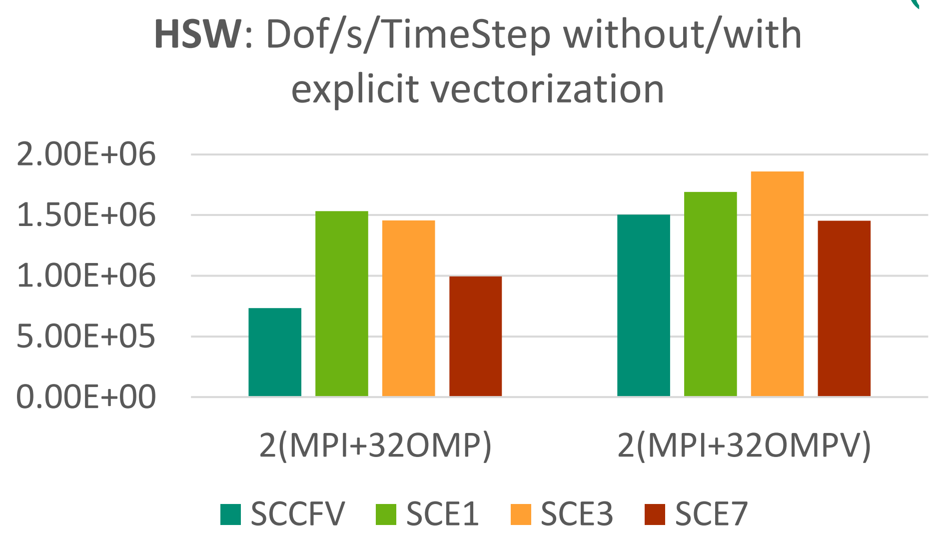
Setup: 128^3 Degrees of Freedom (Dof)

- Wall-clock time over 100 RK44 iterations
- Compare best MPI+OpenMP cases
  - OMP: LayoutRight on Array(cell,spt,**var**)
    - No explicit vectorization
  - OMPV: LayoutLeft on Array(**cell**,spt,var)
    - Explicit vectorization

Results:

- Higher throughput with explicit vectorization
  - LayoutRight performs better in rare cases
    - Better caching?
- Larger throughput at higher orders
  - SCE7 has smaller throughput
- Large benefit in SCCFV
  - High-order improvements are relatively modest





Mach 0.2 Turbulent Flat Plate Boundary Layer

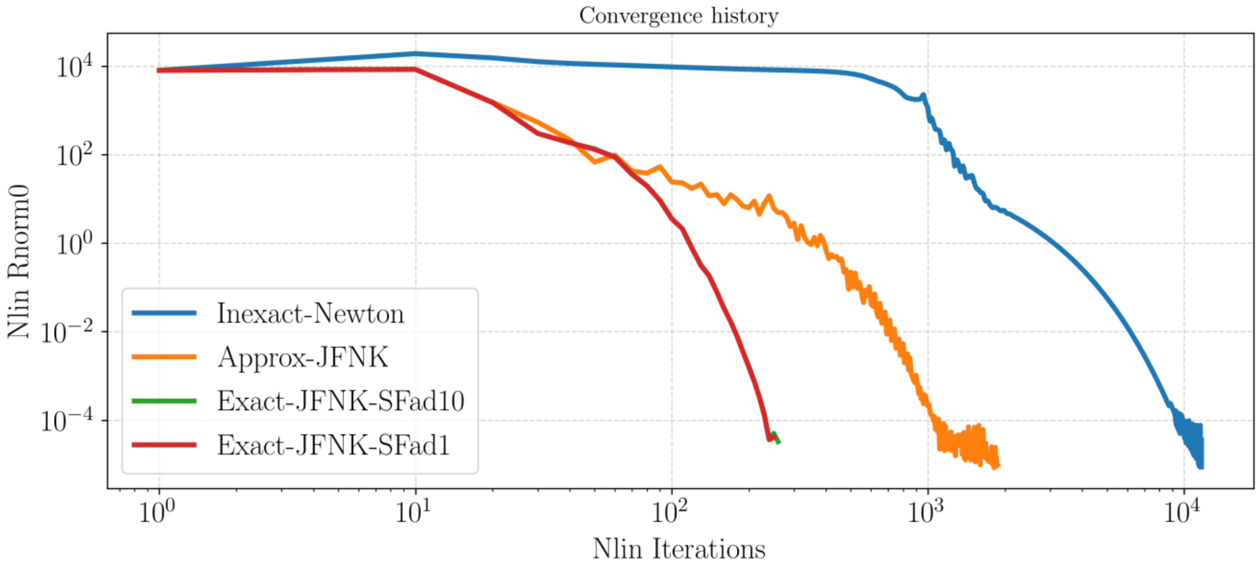
Setup:

- Steady-state: pseudo-transient continuation
- Inexact Jacobian
  - Second-order finite volume discretization
  - First-order inviscid Jacobian, neglect viscous cross-terms
  - Used for inexact Newton and preconditioned JFNK
- Linear solve
  - Block tridiagonal solver or GMRES/ILU

Results:

- Exact matrix-free led to 7x speedup over inexact Newton for SA turbulence model
- Robustness issues when applying matrix-free methods to SST turbulence model

Convergence history for flat plate, Spalart-Allmaras (SA) turbulence model



	Nlin Iterations	Problem Solve Time (s)	Belos Solve Time (s)
Inexact-Newton	11684	109.216	58.7482
Approx-JFNK	1873	51.9301	43.1881
Exact-JFNK-SFad10	262	46.3098	44.2337
Exact-JFNK-SFad1	256	15.2841	13.4616

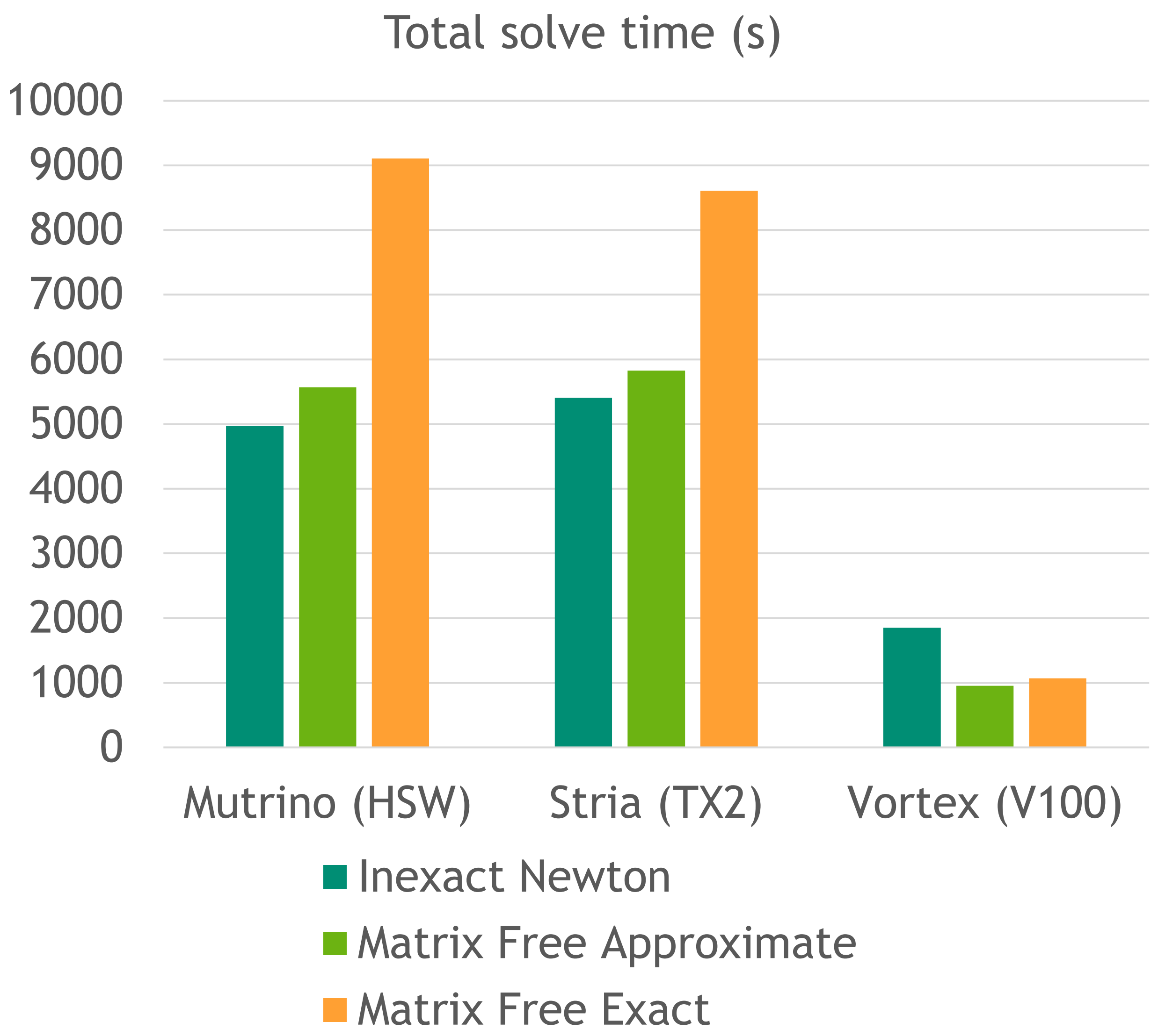
Mach 3.5 Flat Plate Boundary Layer ILES

Setup:

- SCE3, 8 nodes
- Jacobi preconditioner

Results:

- **Inexact Newton** performed better on **CPU** platforms
- **Matrix-free approximate** performed better on **GPU** platforms
- **Matrix-free exact** performed better on platforms with **more threads**
- **Fastest wall-clock time** was on **GPU**
  - **5x** over fastest **HSW** time



Discussion

Discussion

- HPC architectures are changing rapidly which poses a significant challenge
- **Trilinos/Kokkos** offers an efficient way to meet this challenge for large scale, high-fidelity simulations
- High-order and matrix-free methods can improve accuracy while benefitting from the high computational throughput on modern hardware but more R&D is needed to improve robustness and performance for hypersonics

<u>Future Work</u>

- Additional data layout testing (strided or AoSoA)
- Full integration of Kokkos SIMD
- Improve end-to-end performance of implicit solver