# Kokkos in SPARC: Performance Portability Experiences in a CFD Application






*NASA Ames Research Center, April 11th, 2024*

Travis Fisher

Advanced Modeling & Simulation (AMS) Seminar Series

Contributions from: Victor Brunini and rest of SPARC team

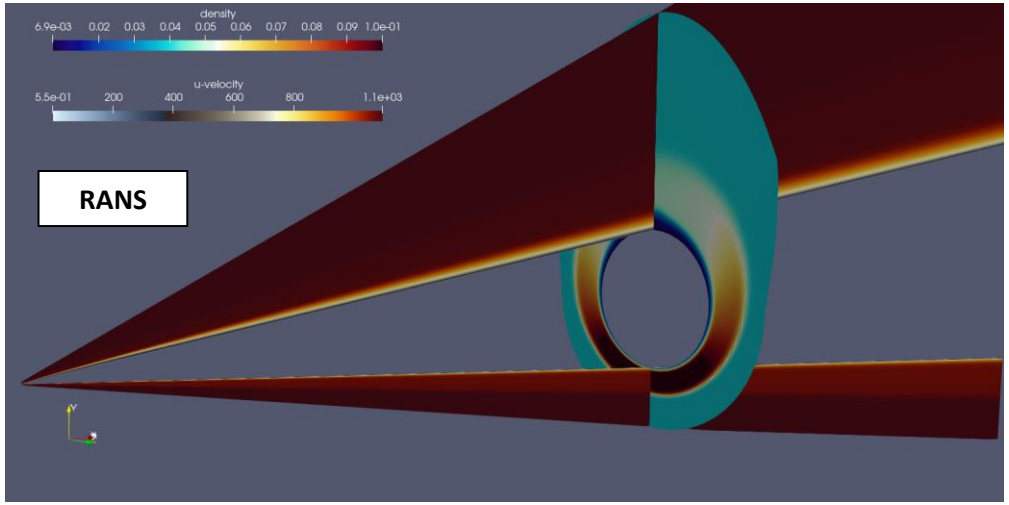
SAND2024-04361PE

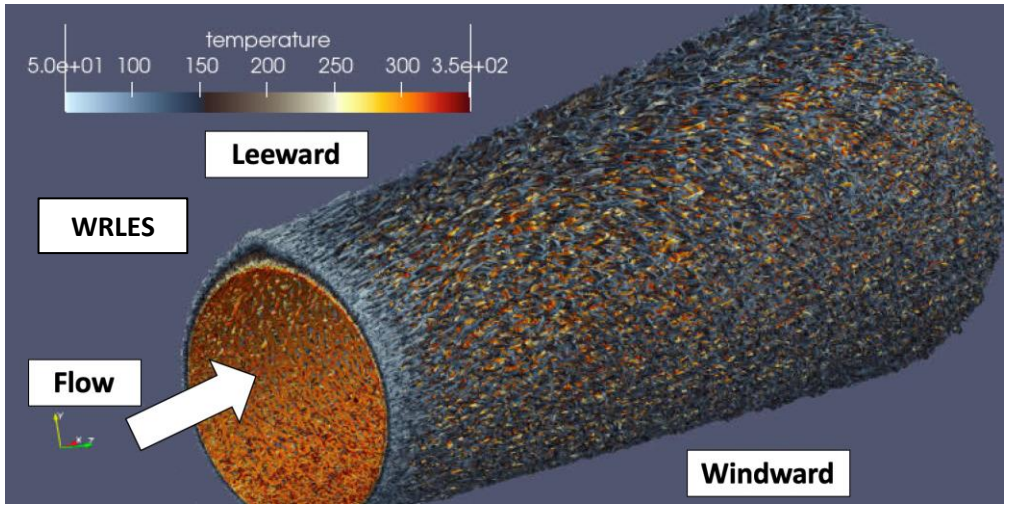
Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

# SPARC Overview

## Goal: Create a credible full-system virtual flight-testing platform for hypersonic vehicles

- Modeling
  - Perfect and non-equilibrium thermal and chemical gas models
  - Euler, Laminar, RANS, Hybrid RANS/LES, LES, and DNS
  - Structured and Unstructured Finite Volume methods
  - R&D in structured and unstructured high-order methods
  - Simulate coupled ablation
  - Couples to SIERRA for full-system thermal and structural analyses
- Performance and Portability
  - Performance Portability through Kokkos
  - Good performance on x86, Arm, and GPU platforms
  - Uses performance portable/scalable linear solvers from Trilinos
- Credibility
  - Validation with UQ against wind tunnel and flight test data
  - Visibility and peer review by external hypersonics community



# Context: SPARC's Performance Portability Business Model

Problem

- SPARC needs to run efficiently on the largest NNSA computers
  - Many factors drive acquisition
  - Use cases will use as much computing power as available
- And commodity hardware
  - Driven more by how many codes run efficiently
  - Run lots of jobs concurrently




Solutions

- Kokkos to limit specialized code
- Performance portable solvers in Trilinos
- Centers of Excellence with vendors to better map abstractions to hardware
- Flexible design and healthy funding

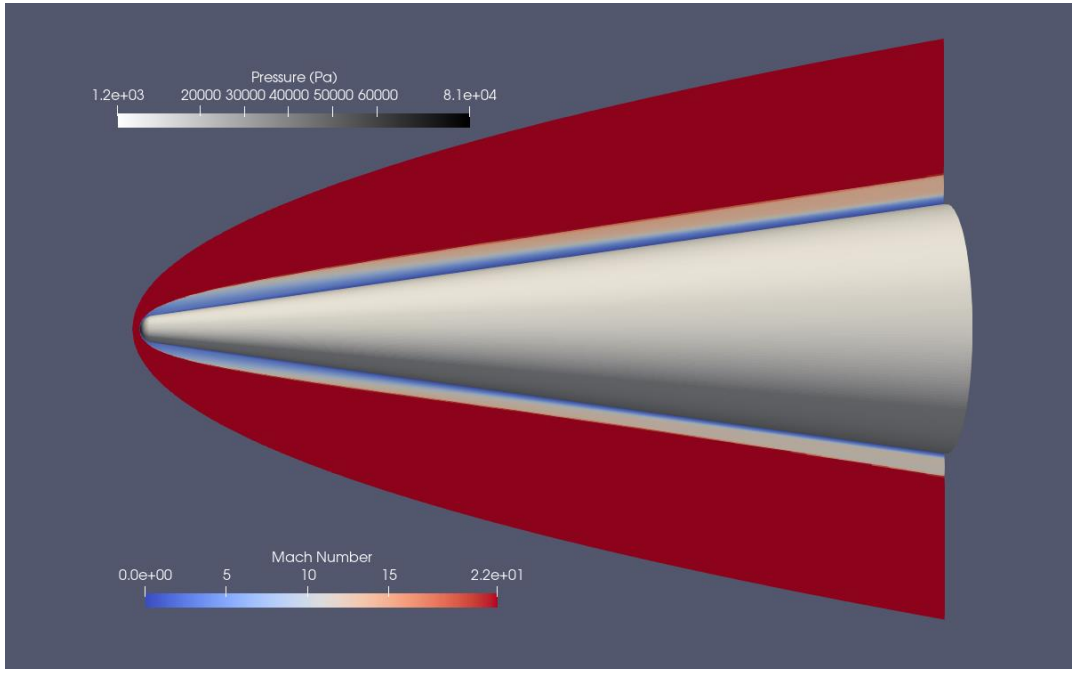




# Performance Portability Results

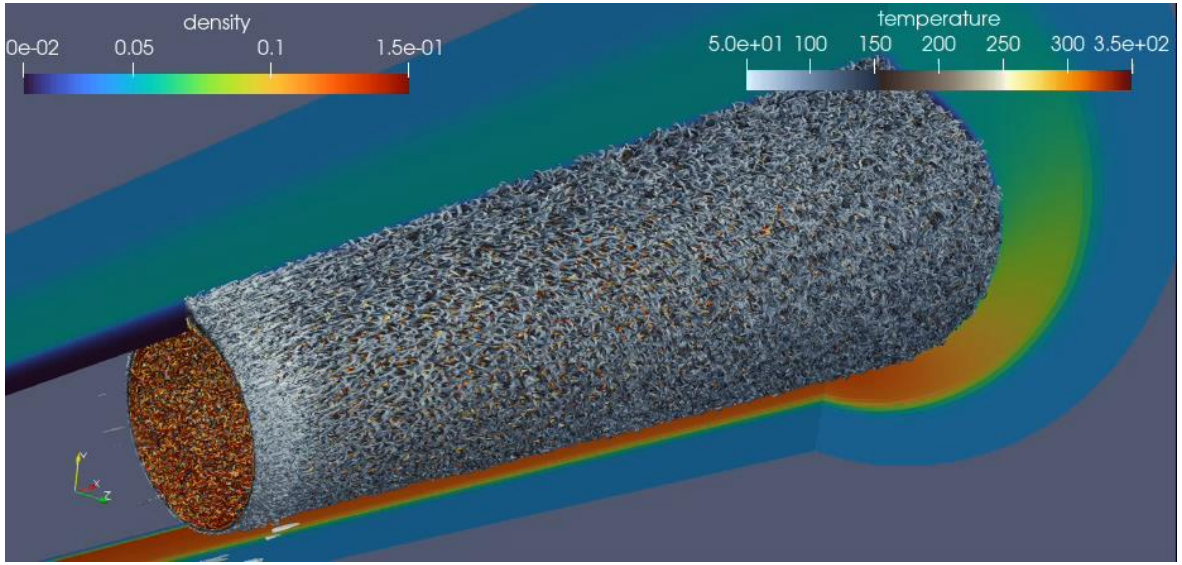
# Problems Evaluated

## Generic Reentry Vehicle (GRV)




- 5 sp/2T reacting gas steady state RANS model
- Representative of trajectory aero-thermal analysis use case

| Mesh | # of cells |
|---|---|
| R0/Seq08 | 4.2M |
| R1/Seq04 | 33.5M |
| R2/Seq02 | 268.4M |
| R3/Seq01 | 2.2B |

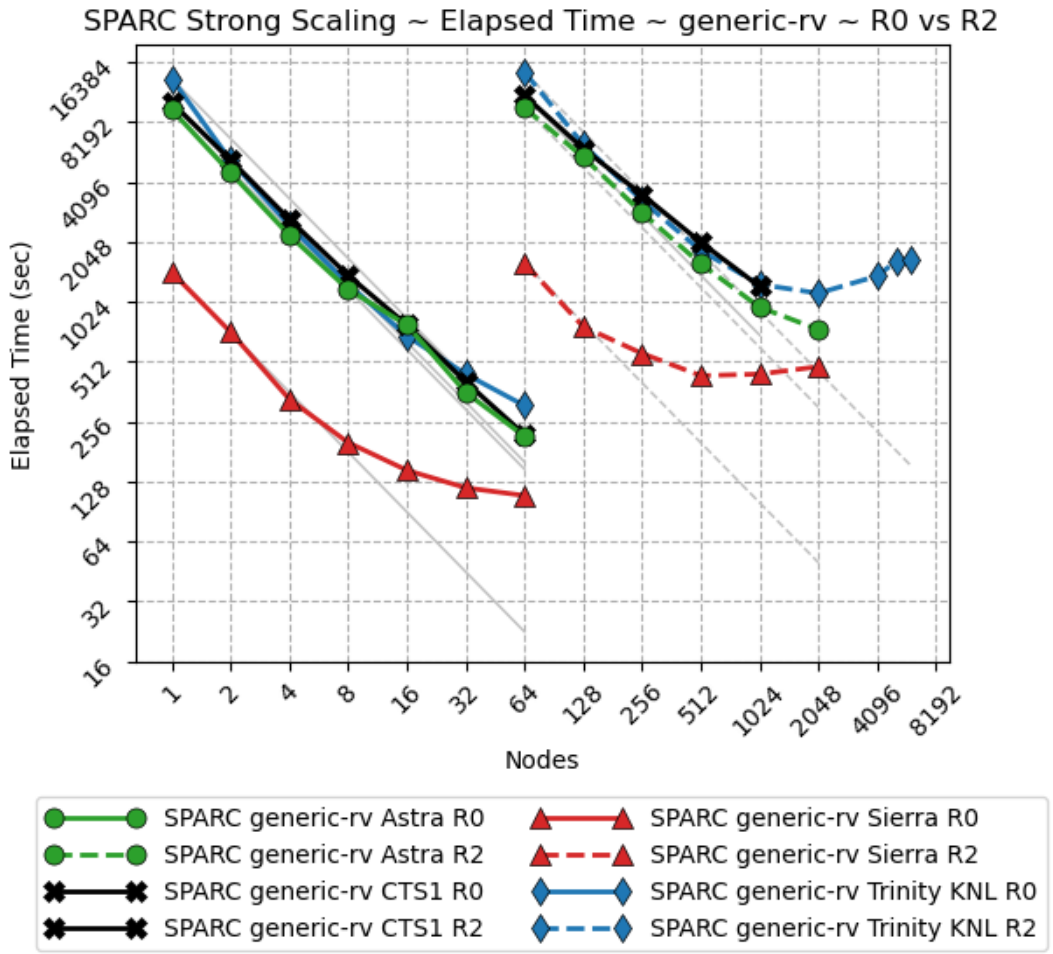


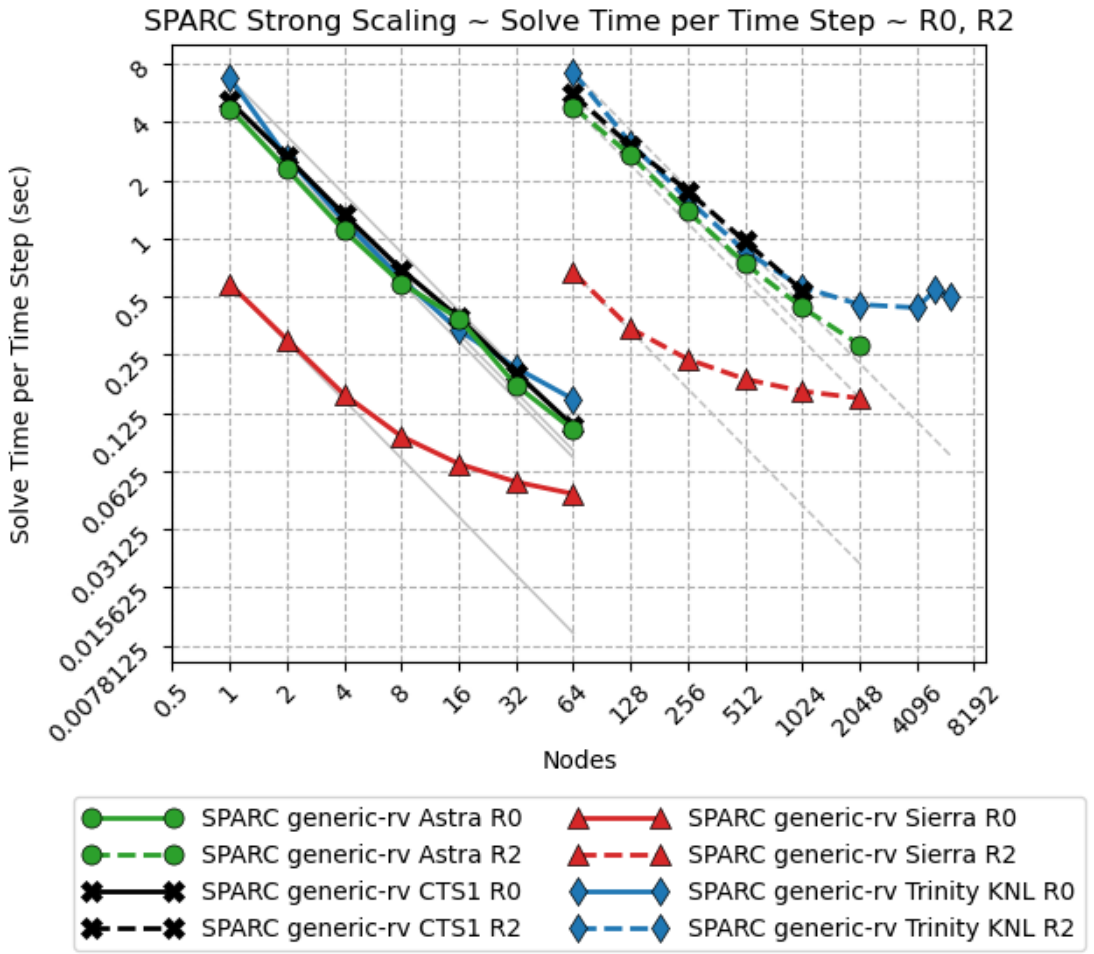

- Perfect gas time accurate LES model
- Exascale driver use case
- Representative of high-fidelity random vibration use case
- 17.3M cell mesh - 11.25 degree section of cone

# Machines Evaluated

| Machine | Node Architecture | FLOPs/Baseline | Bandwidth/Baseline |
|---|---|---|---|
| CTS1 | Intel Broadwell 2x18c | 1.0 | 1.0 |
| ATS1/Trinity | Intel Knights Landing | 1.89 | 2.5 |
| ATS2/Sierra | 4x Nvidia V100 GPU | 20.0 | 25.0 |
| VAN1/Astra | Arm Cavium ThunderX2 | 0.65 | 1.84 |
| ATS4/MI250X (EAS3) | 4x AMD MI250X GPU | 136 | 78.4 |
| ATS3/Crossroads | Intel Sapphire Rapids w/HBM | 6.24 | 12.3 |

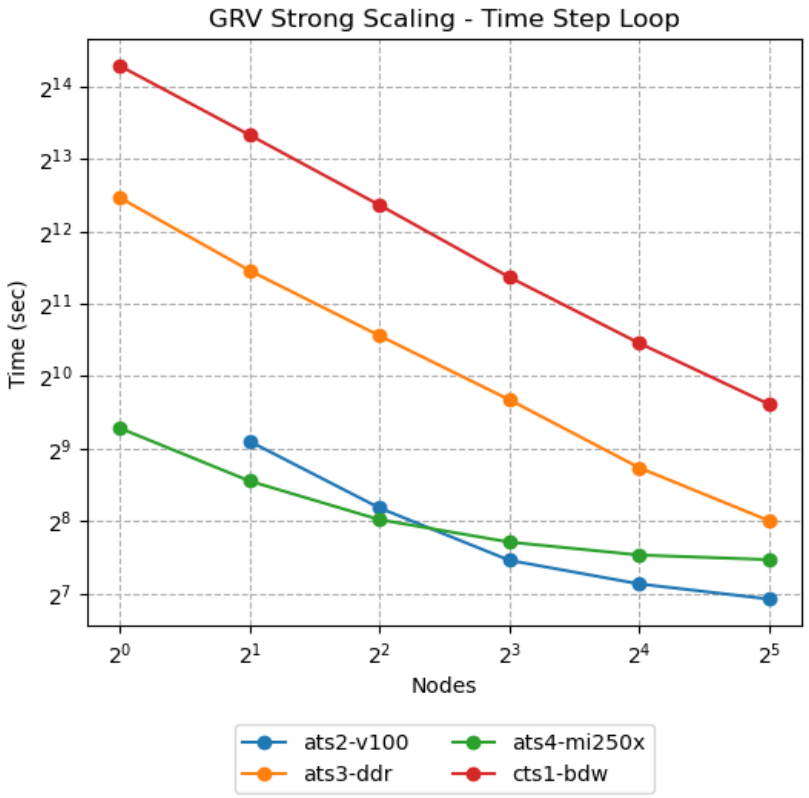
# Generic RV – CPU vs GPU ~2021

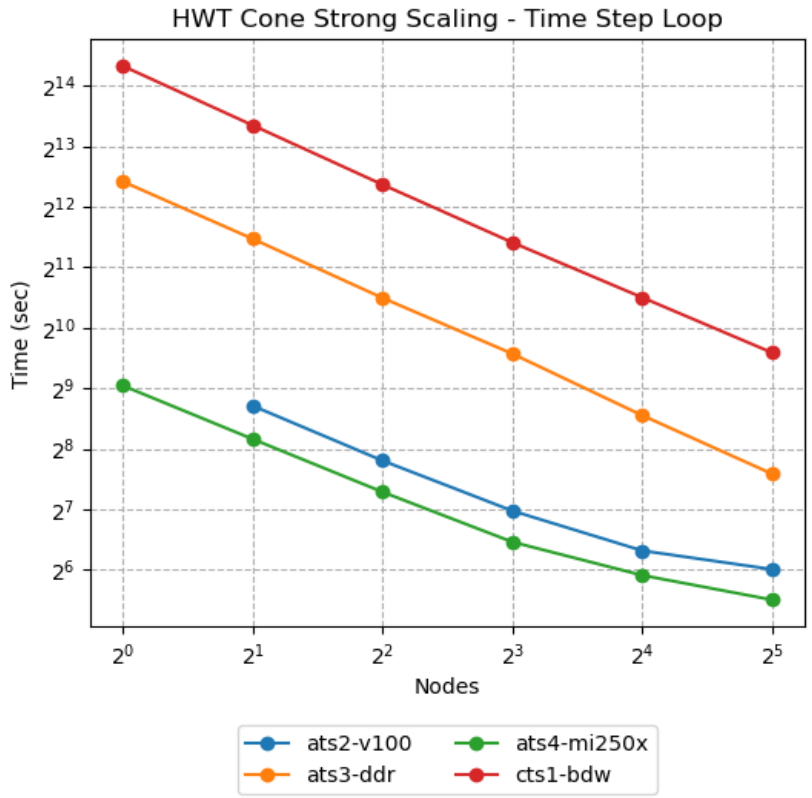



- Achieve 8x speedup moving to V100 GPUs
- Poor scaling of GPU system dominated by linear solver

# Generic RV and HWT Cone – CPUs vs GPUs ~ 2023



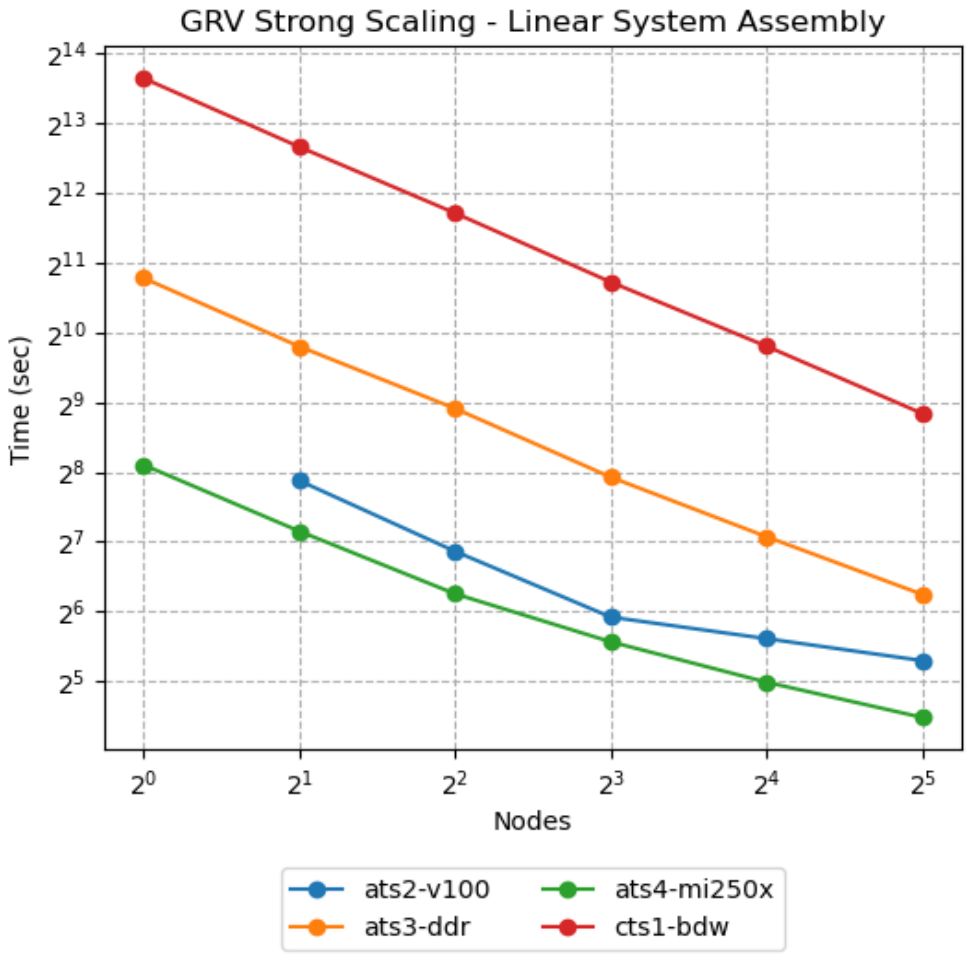

El Capitan early access nodes (ats4-mi250x) show 32x speedup vs Broadwell CPUs

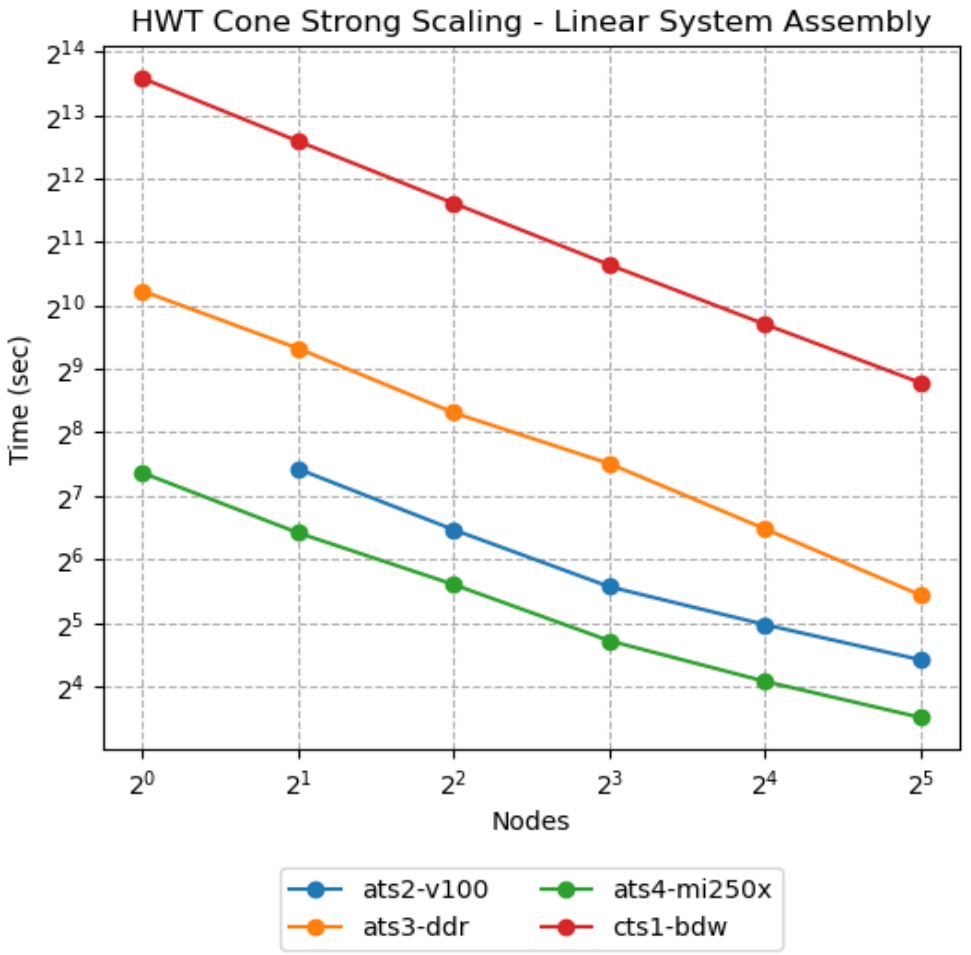
MI250X shows good strong scaling when solver has sufficient parallelism (HWT cone on right)

MI250X speedup modest (40-50%) against V100 GPU

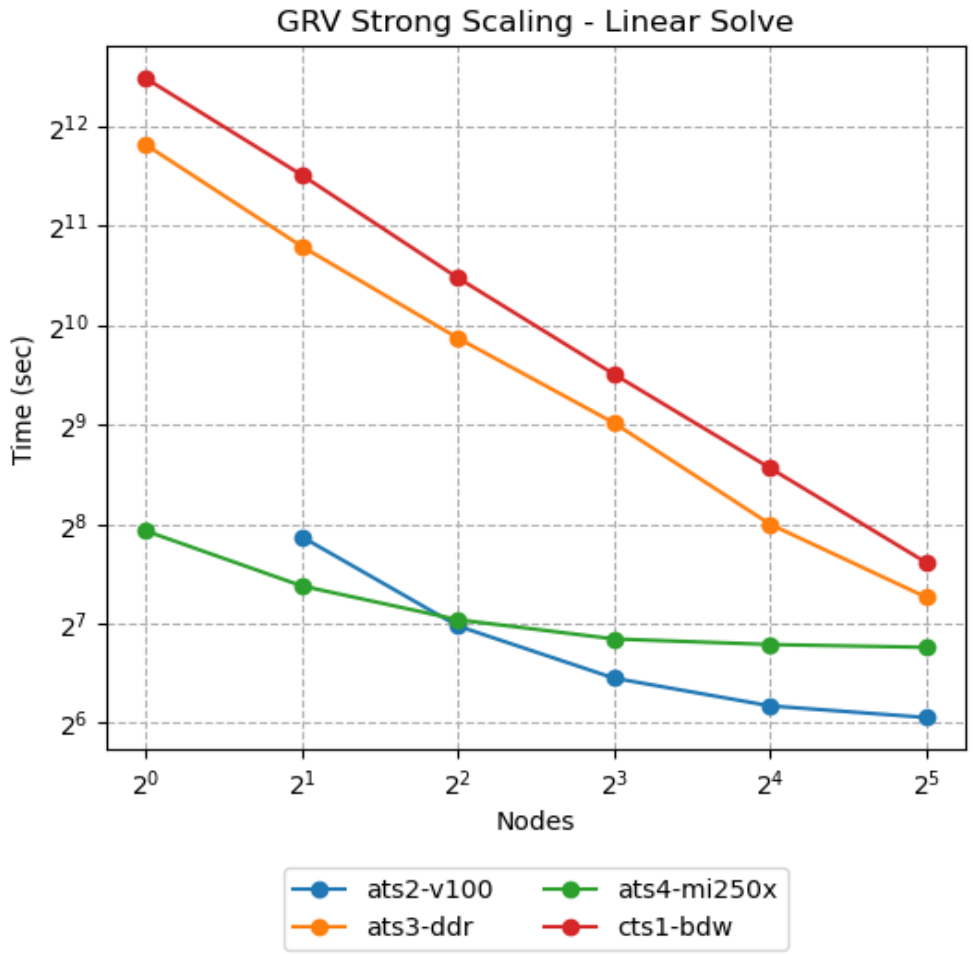
Ignore ATS3 for now (see later slides)

# Generic RV and HWT Cone Linear System Assembly

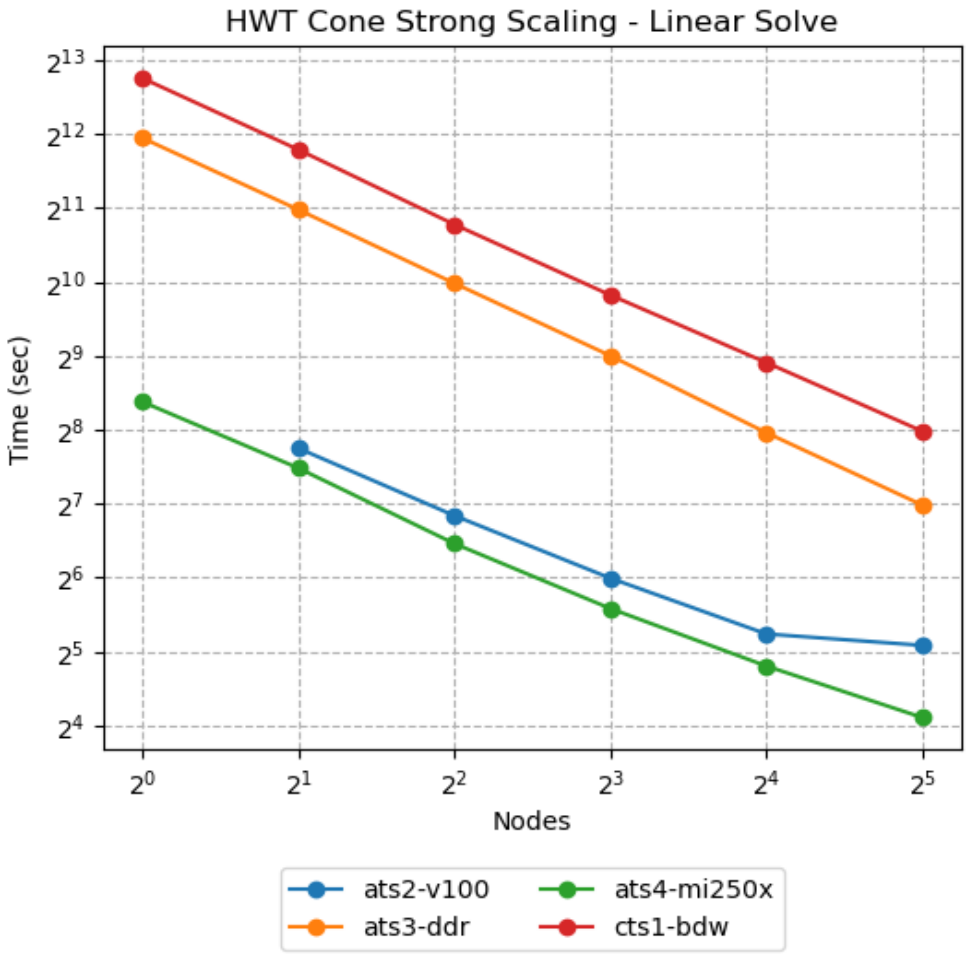


# Generic RV and HWT Cone Linear Solver

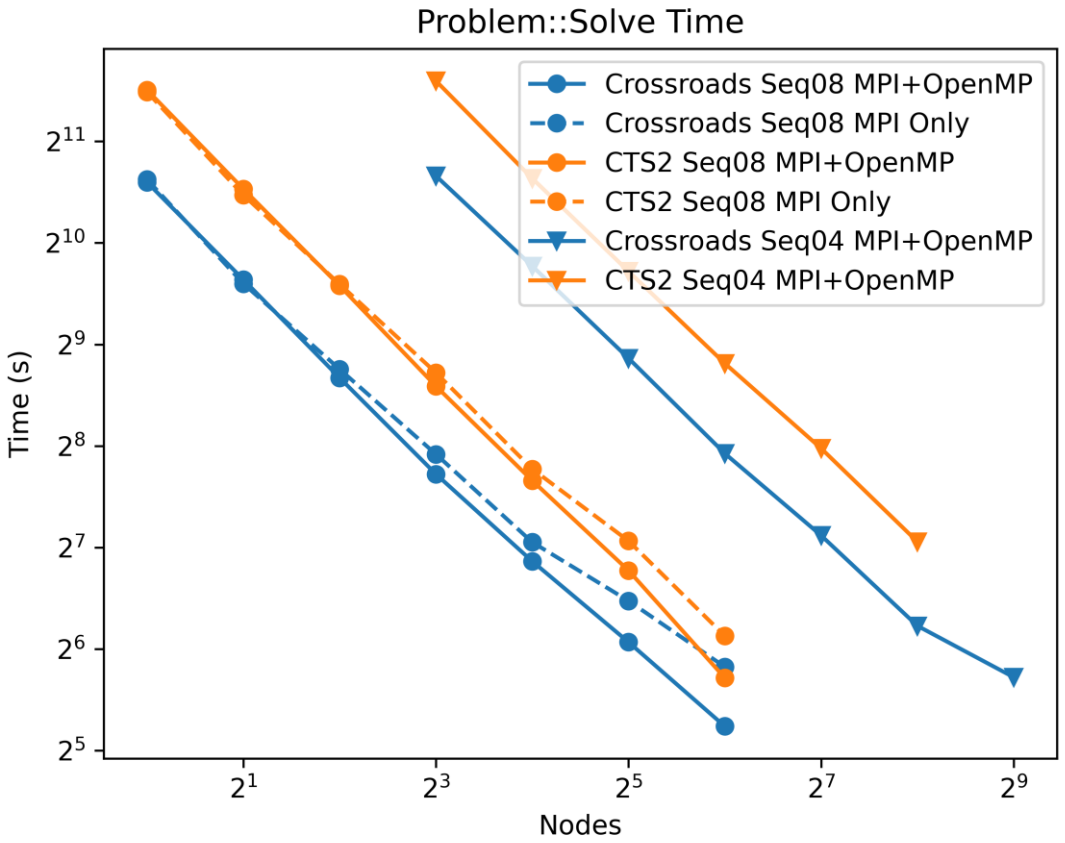


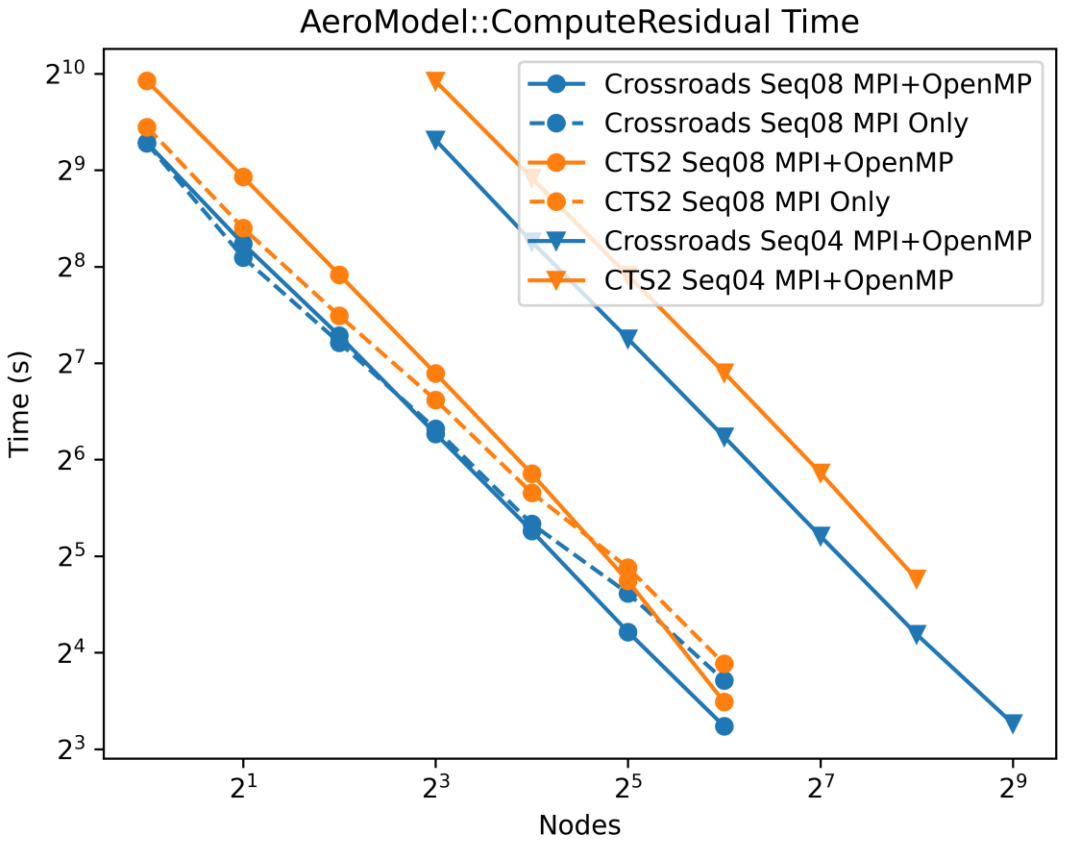

Block Tridiagonal Solver




Block Jacobi Solver

# Generic RV – CPU vs CPU+threads and High Bandwidth Memory



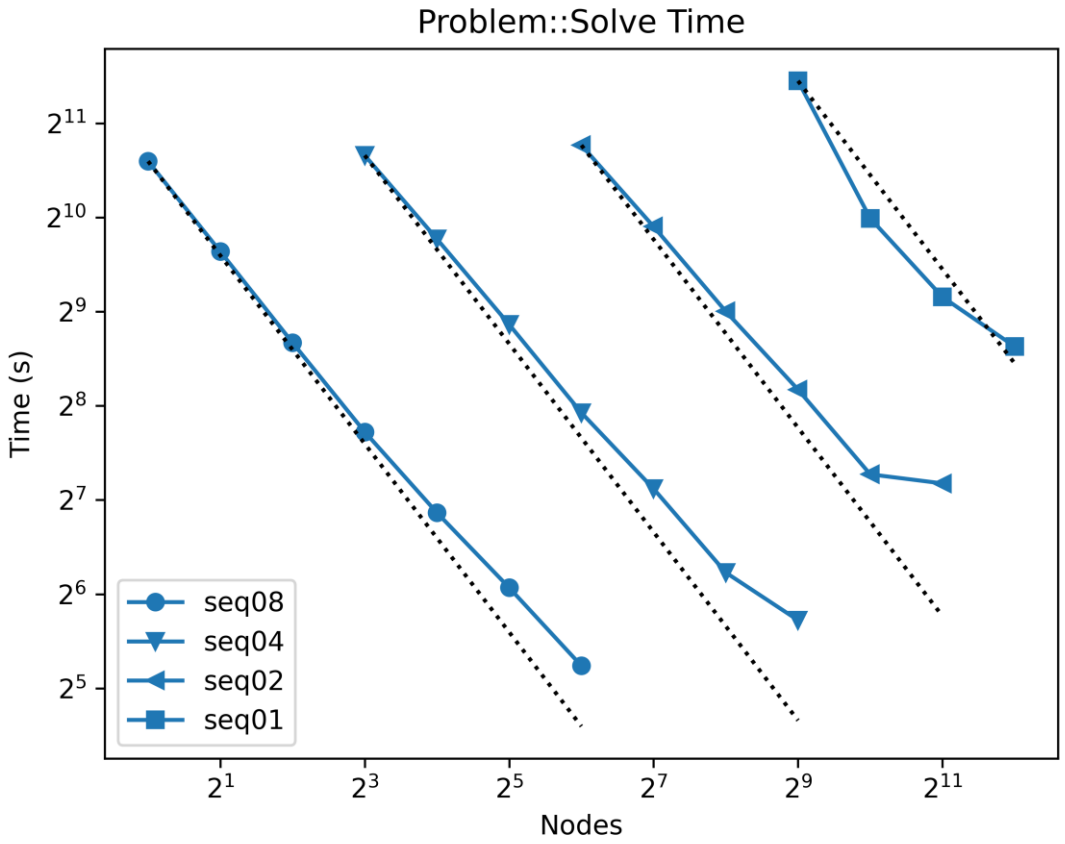

Crossroads has HBM and slingshot interconnect

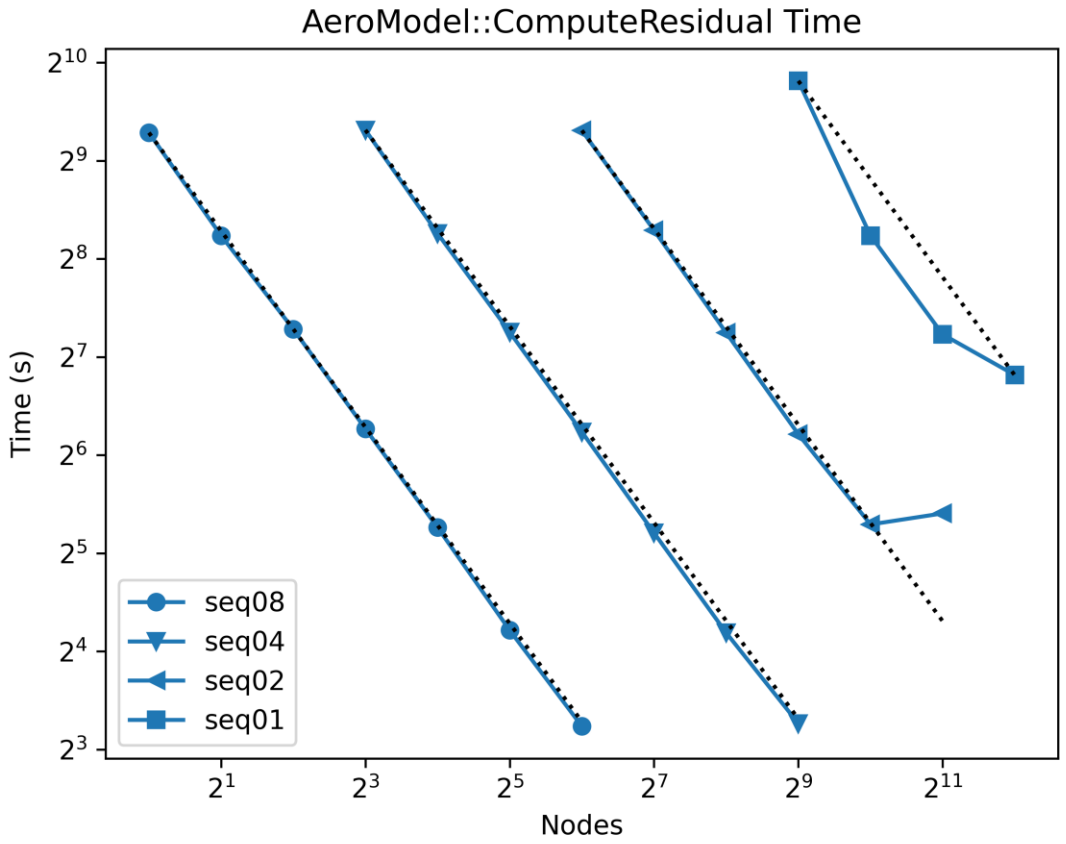
CTS2 has DDR and a different interconnect

Total speedup for crossroads is 40%-90%. 40%-60% faster for residual assembly

MPI+OpenMP is cost competitive and scales better on this architecture

# SPARC Strong and Weak Scaling on Crossroads




Strong scaling shows deviations from ideal

- 2D (line) decomposition becomes limiting at high node counts

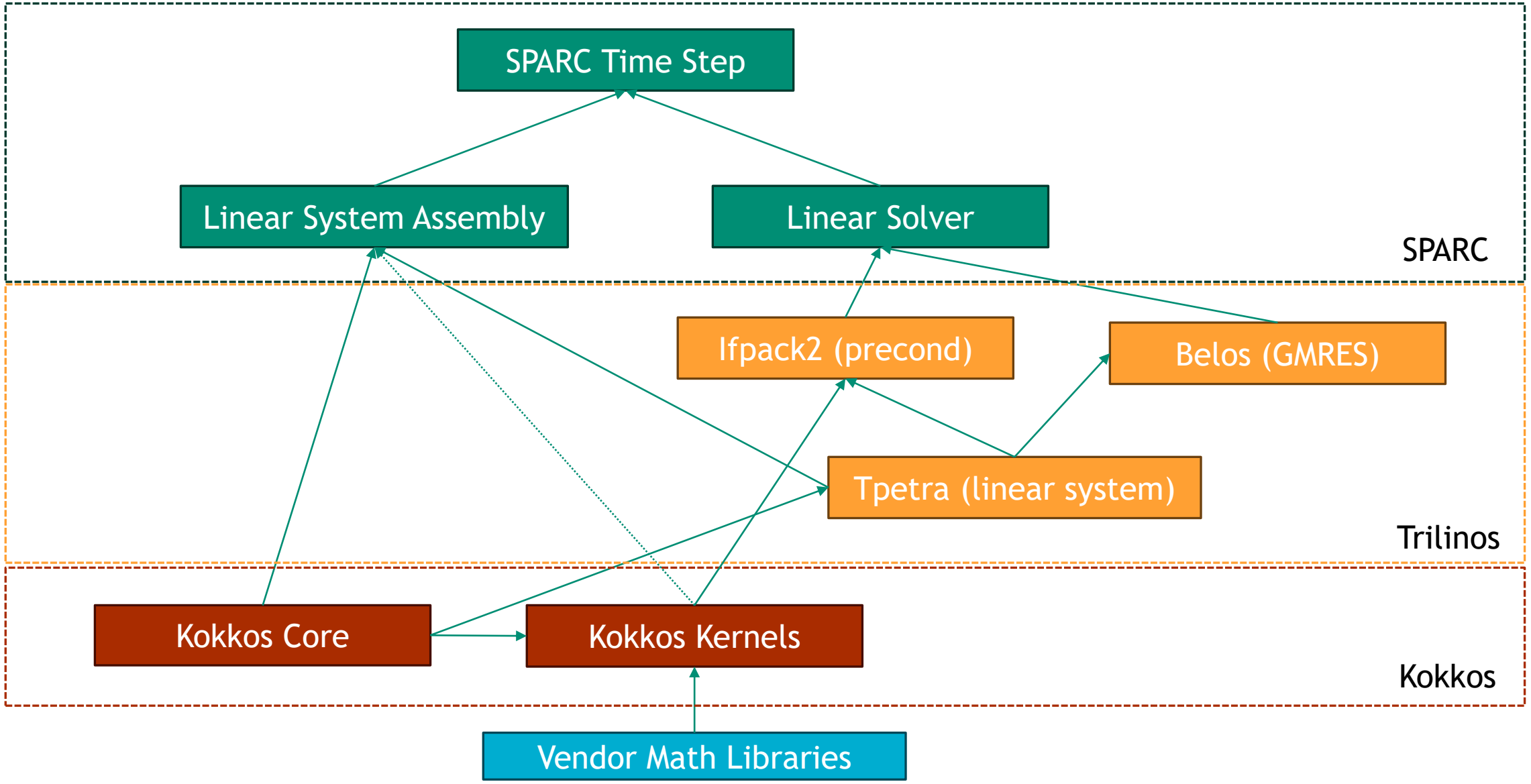
Weak scaling shows issues at highest end

- Possibly running out of memory
- Cannot get memory from rusage without extra effort (reported to intel)

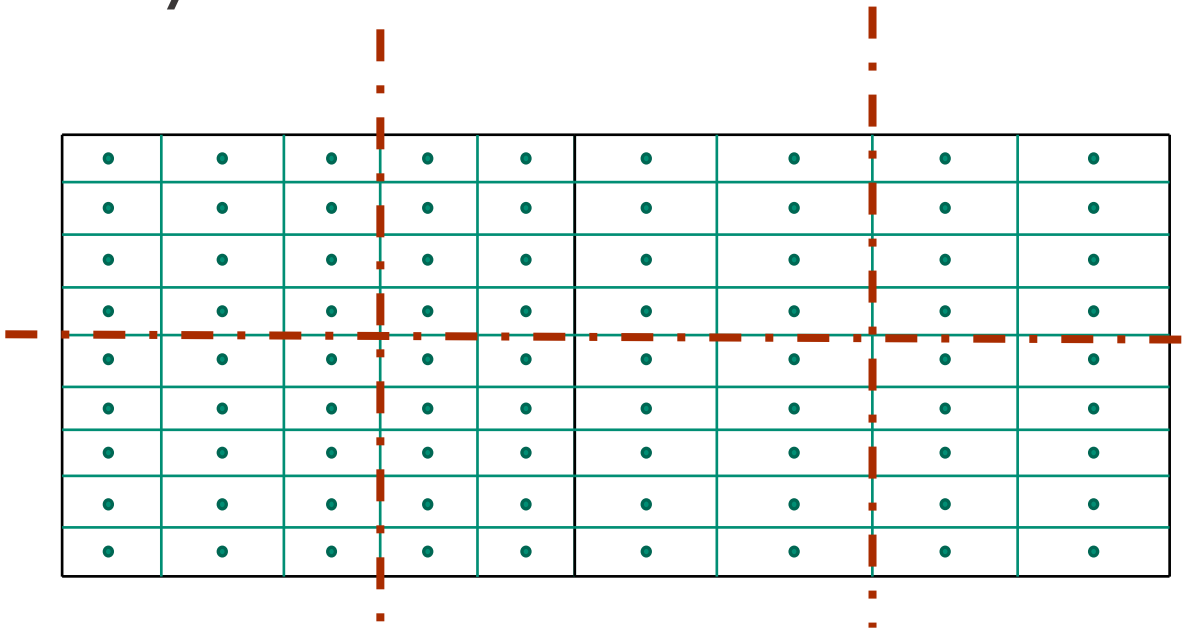


# Performance Portability Technology

# Use of Performance Portable Components



# Parallelism for Linear System Assembly



Expose three levels of parallelism
- Domain decomposition (MPI parallel)
- Cells/faces (thread/vector parallel)
- Equations (thread/vector parallel)

Computer science concerns/challenges
- Loop over the mesh as few times as possible
- Limit GPU memory
- Coalesce memory accesses
- Limit thread divergence
- Register pressure and register spilling for GPU systems
- Stack temporary arrays
- Shared memory/local data storage

Modeling/Productivity concerns
- Different gas models have different numbers of equations
- Different turbulence models have different numbers of equations
- Don't repeat physics code
- Write clear code

```
for (int i = 0; i < neq; ++i)
  f[i] = mass_flux * q[i];
```

# Mapping to Kokkos



Ignore MPI for now

Multi-dimensional looping

- Flatten index space k,j,i <-> r
- SPARC does not currently use Kokkos::MDRangePolicy

Parallel For or Parallel Reduce Execution policy selected per kernel at compile time

- Range policy
  - Attempt to execute over every cell/face at once
  - No shared memory, use stack temporaries
  - No parallelism needed in physics routines
- Team policy
  - Execute over contiguous cell/face groups (team size) per warp
  - Use multiple threads per cell/face
  - Can use shared memory, don't use stack temporaries
  - Pushes parallelism into physics routines
  - Kokkos launch bounds can tell the compiler how much parallelism you're actually going to use

```
using LaunchBounds = Kokkos::LaunchBounds<launch_bounds_num_threads, launch_bounds_min_per_cu>;
using PolicyType = Kokkos::TeamPolicy<ExeSpace, Kokkos::Schedule<…>, LaunchBounds>;
```

# Mapping to Kokkos continued



SIMD

- Can pack coalesced chunks of memory into SIMD Types
- SIMD types use vector instructions on supported platforms (e.g. AVX512)
- Are compatible with other execution policies and modify index space

Atomics vs Coloring

- SPARC maintains the ability to use coloring if needed
- Was helpful on old hardware
- Generally hardware atomics result in much better performance than coloring

Streams

- Execute multiple kernels simultaneously
- Kokkos::Experimental::partition_space
- Use atomics to avoid collisions

# Field Data Structures



SPARC fields

- A container of Kokkos::DualView objects
- Generally one DualView per database block on a given parallel rank
- Rank of dual view depends on discretization (Rank 4 for structured multi-block, Rank 2 for unstructured finite volume and finite element, Rank 3 for DG)

Kokkos::DualView

- SPARC Wraps DualView for convenience

```
double rho = q_dual_view.view<HostSpace>()(k,j,i,0);
```

```
double rho = q_sparc_array(k,j,i,0);
```

- Eliminate explicit Kokkos::subview calls when taking slices

```
auto q_subview =
subview(q_dual_view.view<HostSpace>(),k,j,i, All());
```

```
auto q_slice = q_sparc_array(k,j,i);
```

- Overload indexing to easily manage SIMD field access (access four field entries instead of one)
- We also wrap some helpful host/modify functionality, but host/modify still the biggest source of bugs

Kokkos::View<AnonymousSpace>

- Physics code generally doesn't care where things are executing
- Best not to pass dual views into physics code
- Found a combination of templating and use of Anonymous space helps tremendously with portable physics code

# Temporary Data



We use lots of temporaries

- Normals
- Average states
- Local residual and Jacobian contributions

Where temporaries get allocated depends on the machine and execution policy

- Range Policy: use stack arrays
  - Will spill into global memory on the GPU once you run out of registers
  - Spillage efficiency is dependent on architecture
- Team Policy: Kokkos team shared memory for different device cache levels

Use c++ abstractions and templating to unify physics-level interface

```
auto v = make_temp_array<double[max_compile_time_size]>(allocator, runtime_size);
```

- A stack-type allocator would use the compile time size
- A pool-type allocator would use the runtime size

# Inner loop parallelism

SPARC chooses to abstract inner loop parallelism

Loops
- Looper.flat_for, nested_for, flat_reduce, nested_reduce
- Map to Kokkos parallel_for/parallel_reduce(TeamThreadRange/ThreadVectorRange)

Explicit simultaneous execution
- Looper.run_all
- Kokkos parallel for that executes different code on each thread

Single thread exectuion
- Looper.single maps to Kokkos single(per_thread)
- Avoids contention when modifying single array entries

Synchronization
- Looper.wait_all(futures)
- Does team level synchronization or no synchronization, depending on what's needed.

```
auto r1= looper.flat_for(num_eqns, [&](const int is) { v[is] = 3 * q[is];
});
auto r3 = looper.single([&]() {
  v[3] = q[5];
  q[6] = v[4];
});
auto r4 = looper.run_all(
    [&]() { v[0] = u+c; },
    [&]() { v[1] = u-c; },
    [&]() { v[2] = u; });
looper.wait_all(r1, r3, r4);
double vel_squared = 0;
looper.flat_reduce(3, [&](const int d, double &accum) { accum += v[d] *
v[d]; }, Kokkos::Sum<double>(v_squared));
```



# Developer Productivity

# Some observations

Not everything needs to go on the GPU
- Everything performance critical (linear system assemble and linear solver) does
- Pre and post processing depends on need
- Good timers can help (see LLNL caliper for example)

Know your development community
- Seems easier to train (motivated) physics experts in parallel programming than the other way
- Train people to understand concepts and get used to looking at code
- Use abstractions to make code look more familiar and hide complexity

Recognize writing performance portable code is more complex and time consuming
- Build times on some systems may be long
- Build errors may be difficult to understand and trace – use of an IDE very helpful
- Code design needs different considerations
- Can be helpful to tailor the build system to your needs. E.g. CMake code generation

Transform a kernel at a time
- Establish CPU correctness
- Establish GPU correctness
- Establish GPU performance
- Check CPU performance

# Some observations continued



Performance portability investment pays off

- Nvidia GPU work was very time consuming
- AMD GPU work went very quickly
- Nvidia GPU work helped with explicit SIMD
- SIMD work helped with hierarchical parallelism
- Having an OpenMP backend finally paid off for Sapphire Rapids

Be wary of advanced vendor portability features

- Avoid Unified Virtual Memory
- Shared libraries can be alternative to Relocatable Device Code
- The Kokkos team and user community may have good advice for your situation



# Questions?