SAND2018-7188C

Employing Multiple Levels of Parallelism for CFD at Large Scales on Next Generation High-Performance Computing Platforms

Micah Howard, Travis Fisher, Mark Hoemmen, Derek Dinzl, James Overfelt, Andrew Bradley, Kyungjoo Kim, Siva Rajamanickam

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.

INTRODUCTION

Architectures for high performance computing (HPC) are shifting to higher degrees of parallelism on-node

- Evidence of this: Intel Xeon Phi, Nvidia GPU

Why is this being done? Why is this important?

As architectures get more exotic this brings about more application code complexity

This work and this presentation is entirely focused on CFD in the context of high performance computing

- No discussion of aerodynamic flow for our use-cases – this is intentionally nondescript

MOTIVATION

Trinity at Los Alamos National Lab, USA

- Phase I: Intel Xeon Haswell nodes (~10 petaflop/s)
  - 32 cores, 2 threads/core, 90 GB/s DDR bandwidth, ~360W/node TDP
- Phase II: Intel Xeon Knights Landing nodes (~30 petaflop/s)
  - 68 cores, 4 threads/core, wider vector units, 90 GB/s DDR bandwidth, 480 GB/s on-package memory bandwidth, ~260W/node TDP

MOTIVATION

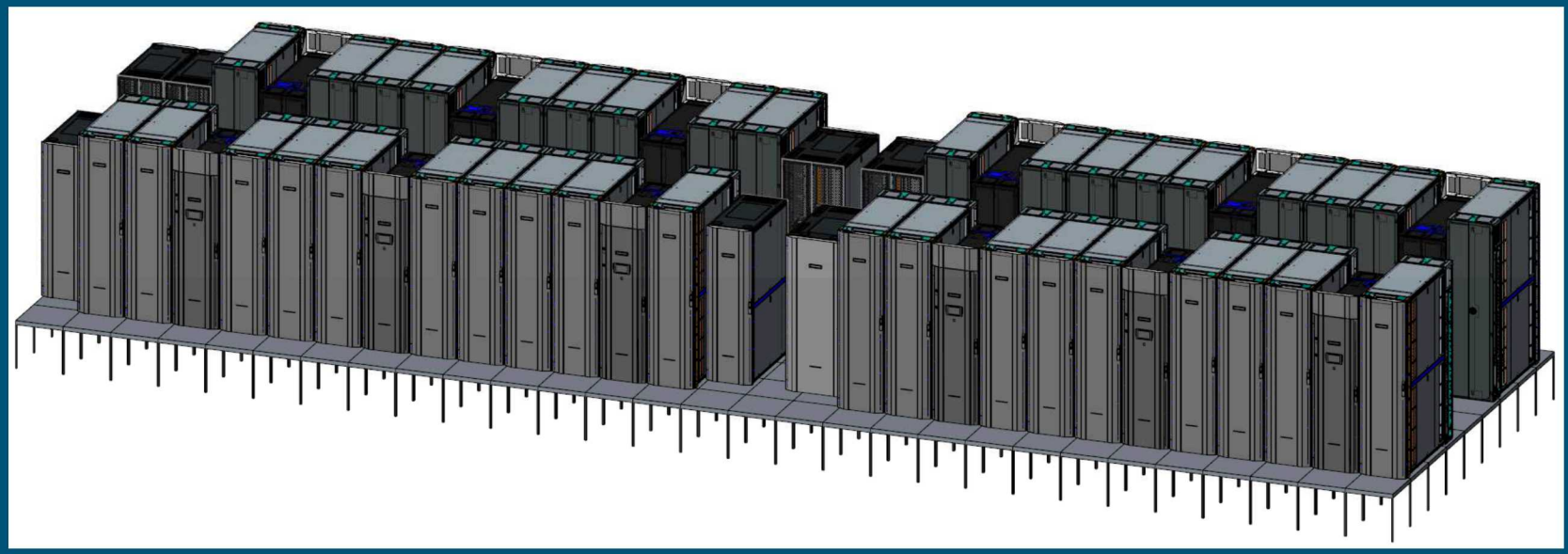
Sierra at Lawrence Livermore National Lab, USA

- IBM Power9/Volta 100 nodes (~125 petaflop/s aggregate)
  - Power9: 44 cores, 8 threads/core, 90 GB/s DDR bandwidth
  - Volta: ~900 GB/s on-package memory bandwidth, ~160W/GPU TDP

MOTIVATION

Astra at Sandia National Lab, USA

- HPE ARM Thunder X2 nodes (~2 petaflop/s aggregate)
  - 56 cores, higher bandwidth on-package memory, unknown TDP



PERFORMANCE PORTABILITY

What is performance portability?

Why should we care about performance portability?

Why should you care about performance portability?

What are the challenges and what are our options?

- Disparate set of architectures and an unknown future
- The challenge of heterogeneous systems (like ATS-2)
- Many programming models
- Separation of data layout from the programming models

KOKKOS and SPARC

Kokkos – Sandia developed programming model

- github.com/kokkos/kokkos
- Combines data layout with portable parallel constructs (parallel_for, parallel_reduce)
- Backends for: serial execution, OpenMP, CUDA, ROCm, etc.

Sandia Parallel Aerodynamics and Reentry Code (SPARC)

- Solves compressible Navier-Stokes equations
- Perfect and reacting gas models
- Laminar and RANS turbulence models
- Primary discretization is cell-centered finite volume
- Research on high-order finite difference and discontinuous Galerkin discretizations
- Uses Kokkos for performance portable implementation
- Goal: solve aerodynamics problems for Sandia on 'leadership' class supercomputers

CORE DESIGN ELEMENTS: DATA LAYOUT

Data layout has a huge effect on architecture performance

Roughly speaking: row-major for CPUs and column-major for GPUs

'functor' construct to implement the loop body of a parallel_for

```
void operator() (int k, int j, int i) const;
```

CORE DESIGN ELEMENTS: FUNCTION DISPATCH

Function dispatch selects between run-time (dynamic) and compile-time (static) invocation

GPU requires static invocation (in our implementation)

CPU has a mix of dynamic and static invocation (balance compile time and run-time performance)

```
template <bool is_dynamic> struct Dispatcher
{
  template <typename Type>
  static void computeFlux (Type* object) {
     object->computeFlux();
  }
};
```

```
Dispatcher<true>::computeFlux<Flux>(object);
```

CORE DESIGN ELEMENTS: KERNELS

Kernels apply a parallel construct over a mesh to do work

```
template <typename Scalar, typename GasModelType, typename KokkosExeSpace>
class ComputeResidualVolumeKernel :
   public GasModelKernel<Scalar, GasModelType, KokkosExeSpace>,
   public MeshTraverserKernel<ComputeResidualVolumeKernel<Scalar, GasModelType,
       KokkosExeSpace>, KokkosExeSpace>
{
private:
   typedef MeshTraverserKernel<ComputeResidualVolumeKernel<Scalar, GasModelType,
       KokkosExeSpace>, KokkosExeSpace> MeshTraverser;
   const Array4D<Scalar> cell_primitive_vars;
   ...

public:
   ComputeResidualVolumeKernel(const StructuredBlock& blk, const
       MaterialCompFluid<Scalar>& gasmodel, const Array4D<Scalar>& cell_V, ...) :
      GasModelKernel<Scalar, KokkosExeSpace, GasModelType> (gasmodel),
      MeshTraverser(blk.kmin, blk.kmax, blk.jmin, blk.jmax, blk.imin, blk.imax),
      cell_primitive_vars(cell_V),
      ...
   { /* constructor code */ }

   KOKKOS_FORCEINLINE_FUNCTION
   void compute(const Index& k, const Index& j, const Index& i) const
   { /* kernel functor code */ }

   void Run() const
   {
      this->MeshTraverser::Run(); // traverse through the current mesh block,
          running the above kernel functor code for each mesh entity
   }
}
```

CORE DESIGN ELEMENTS: LINEAR SOLVERS

Performance portable linear solvers are immensely important

- Solve typically takes ~50% of execution time

SPARC uses many solvers from the Trilinos package

- github.com/Trilinos/Trilinos
- Primary solvers for hypersonics are block Jacobi and block Tri-Diagonal solvers

KokkosKernels provides threaded/vectorized versions of these

- SIMD vectorization achieved through compact BLAS layout and operations

PERFORMANCE ANALYSIS: INTRODUCTION

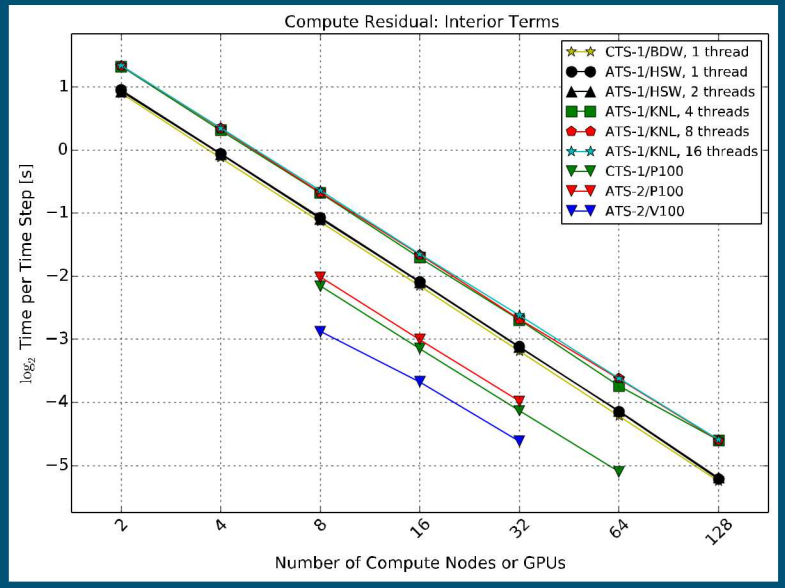
Performance analysis on 3 primary architecture types:

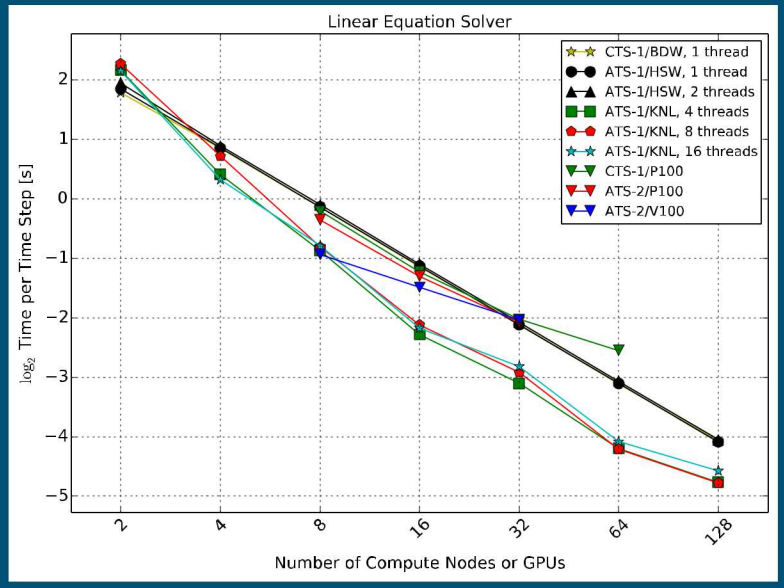
- Intel Xeon systems
  - Haswell nodes – ATS-1/HSW (Trinity)
  - Broadwell nodes – CTS-1/BDW ('commodity cluster')
- Intel Xeon Phi
  - Knight's Landing nodes – ATS-1/KNL (Trinity)
- Nvidia GPU
  - Broadwell/Pascal 100 nodes – CTS-1/P100 (GPU testbed)
  - Power8/Pascal 100 nodes – ATS-2/P100 (ATS-2 early-access testbed)
  - Power9/Volta 100 nodes – ATS-2/V100 (ATS-2 look-alike testbed)

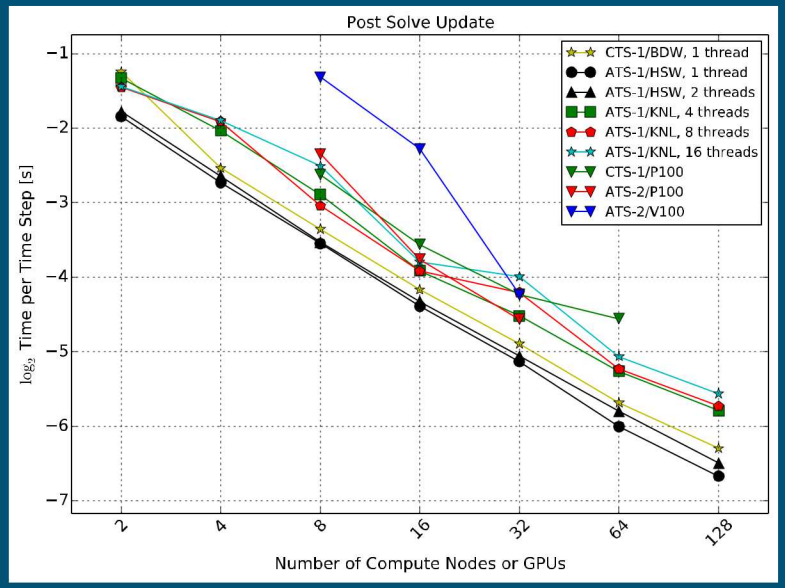
2 primary cases:

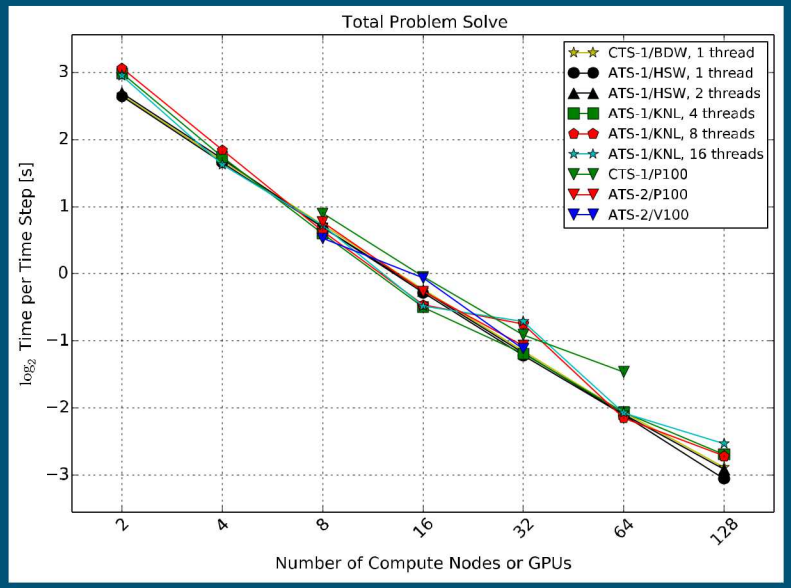
- Perfect gas: ~Mach 6 flow around a sphere-cone geometry
- Reacting gas: ~Mach 9 flow around same geometry with 5-species ari model

PERFECT GAS: STRONG SCALING

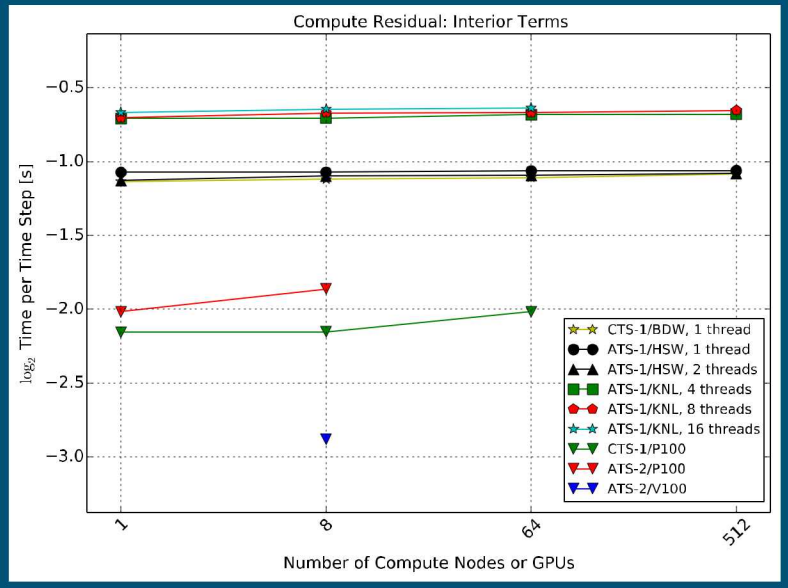


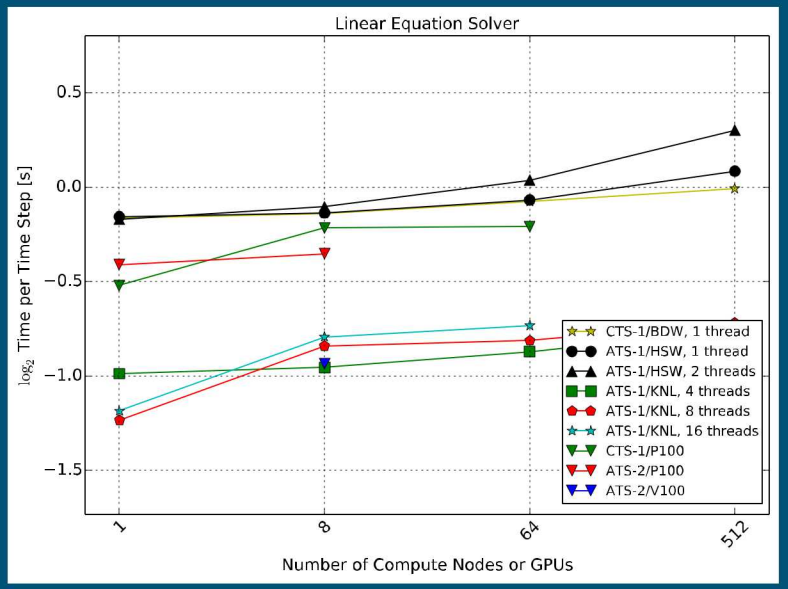


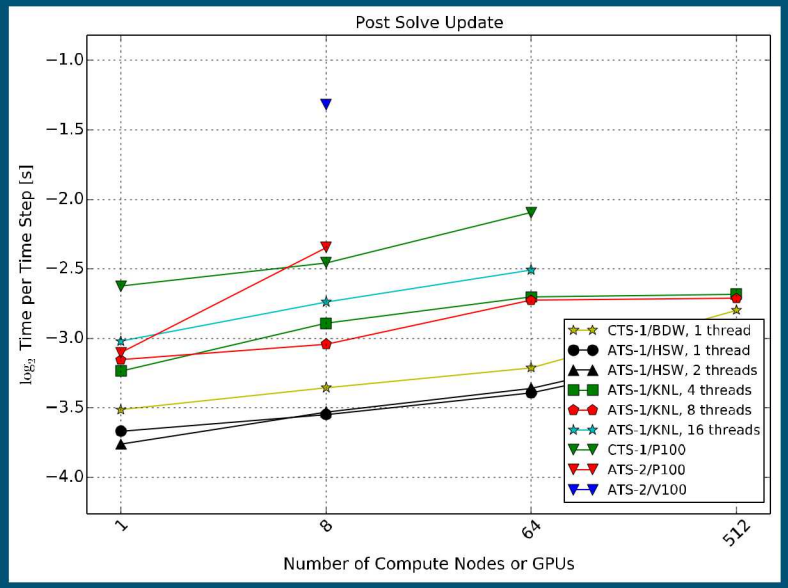


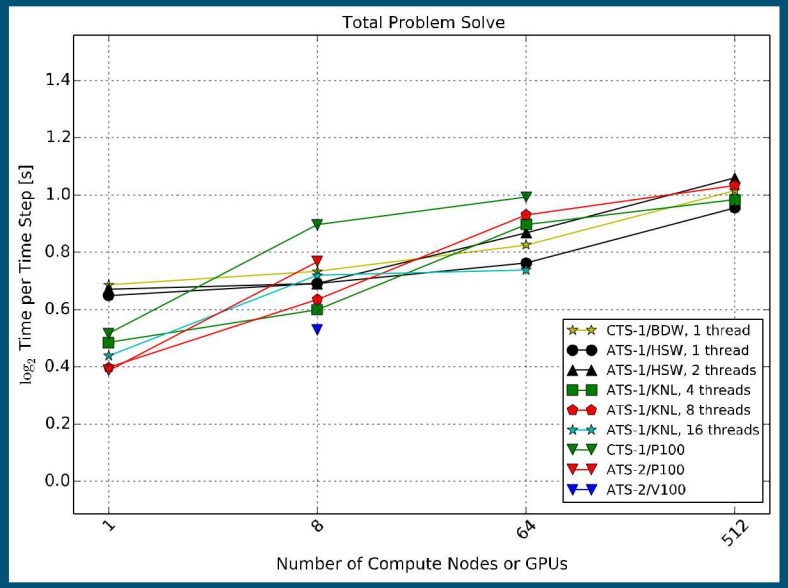


PERFECT GAS: WEAK SCALING

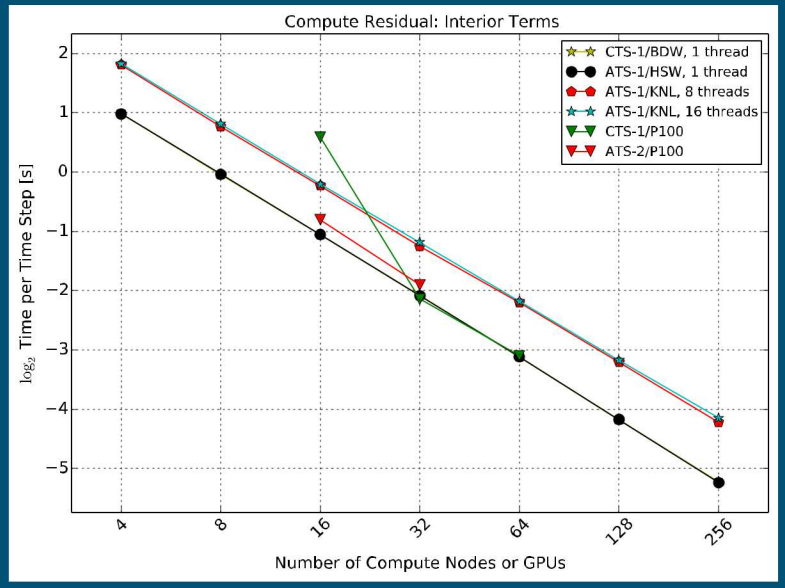


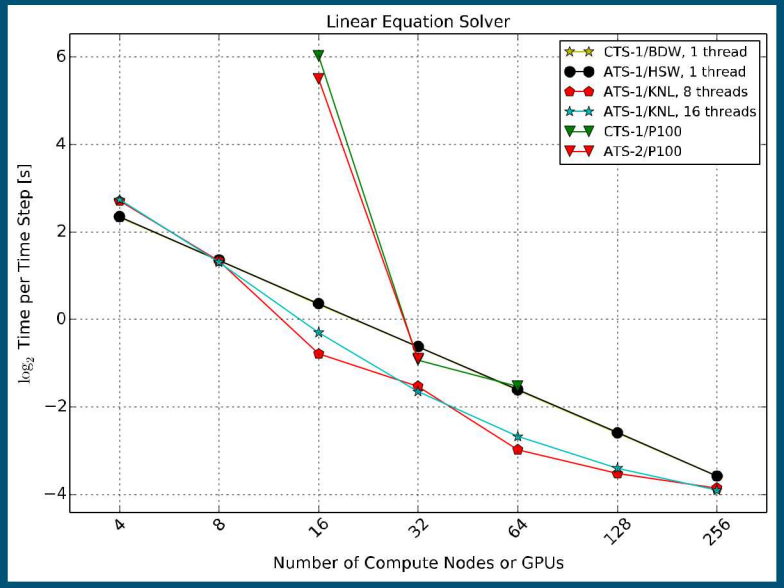


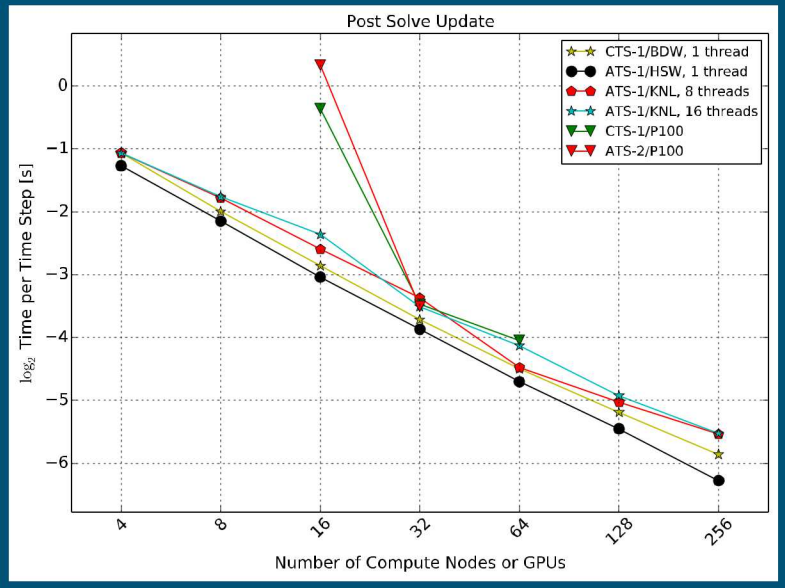


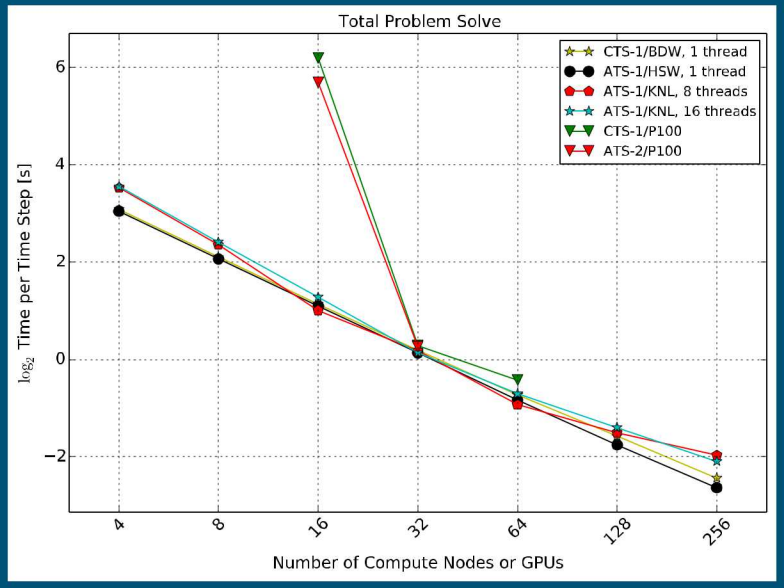


REACTING GAS: STRONG SCALING

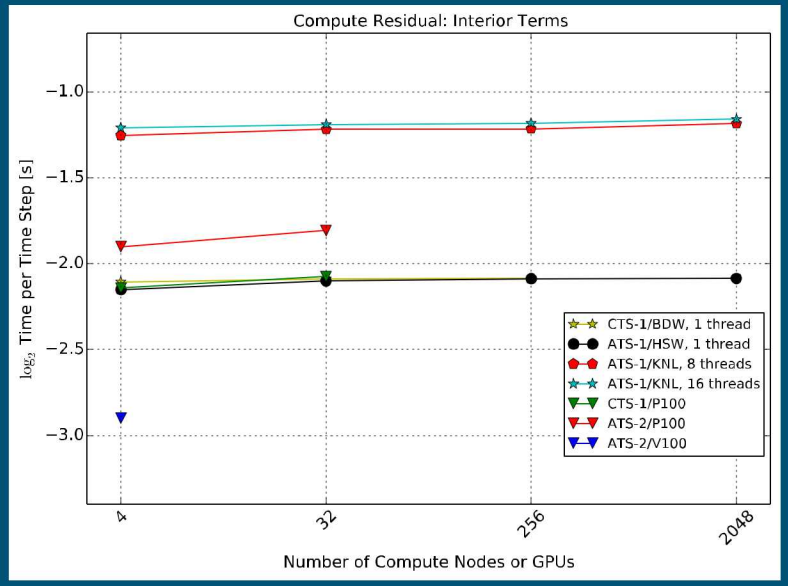


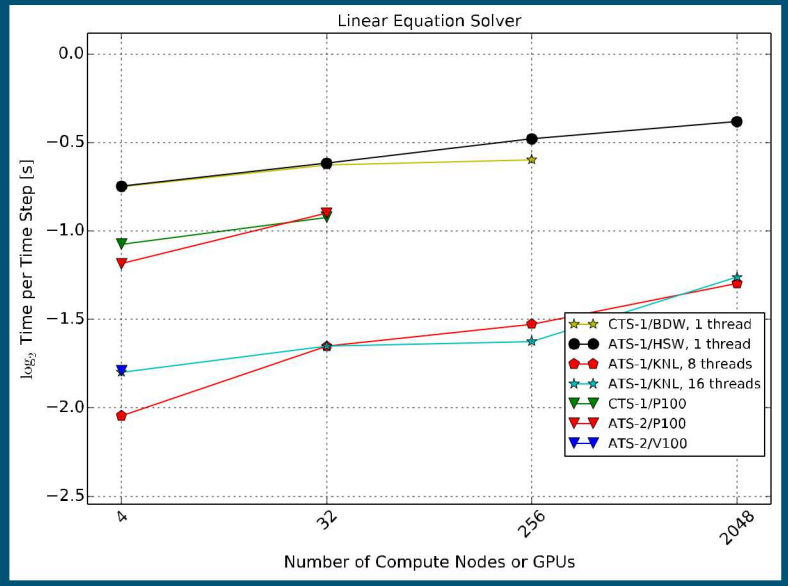


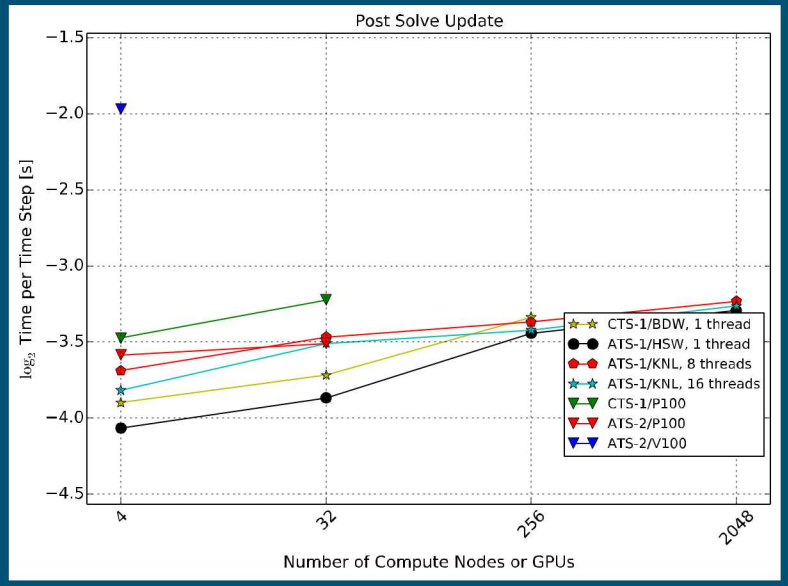


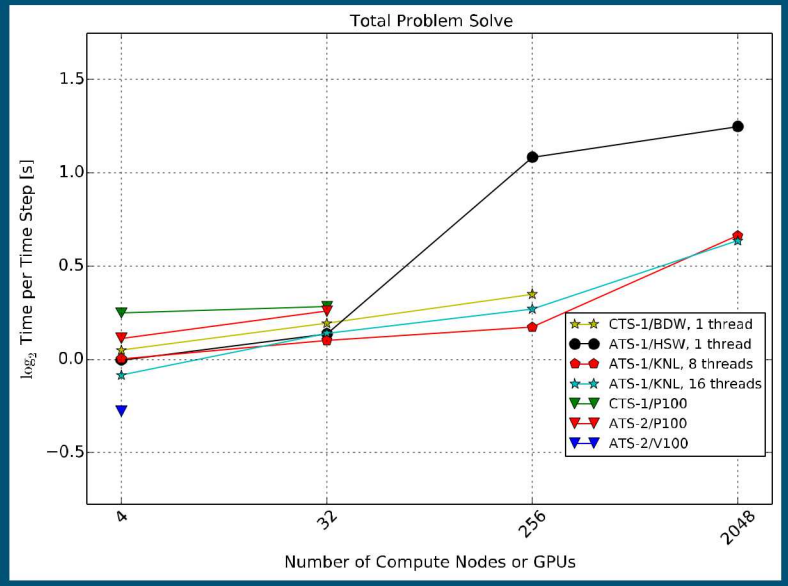


REACTING GAS: WEAK SCALING









CONCLUSIONS AND ON-GOING WORK

SPARC is a single CFD code base that is running efficiently on multiple architectures

Built on Kokkos for performance portability

Accomplishments:

- All computational functions have been implemented as Kokkos kernels
- Solver performance on KNL is ~2x faster than CPU
- Assembly performance on P100/V100 is ~2-4x faster than CPU/KNL

On-going work:

- Optimizing assembly on KNL via SIMD vectorization
- Optimizing solver performance for GPU

QUESTIONS