This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

SAND2020-8715C

Experiences incrementally porting a large legacy finite element application to Sierra using Kokkos

PRESENTED BY

Victor Brunini

Jon Clausen, Mark Hoemmen, Alec Kucala, Malachi Phillips

SAND2020-8555 C

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.

Sandia Nationa
laboratory mar
Technology &
LLC, a wholly
International I
Energy's
Administration

1. What is Aria?
2. History of Aria Performance Portability Work
3. Current performance results
4. Lessons Learned

What is Aria?

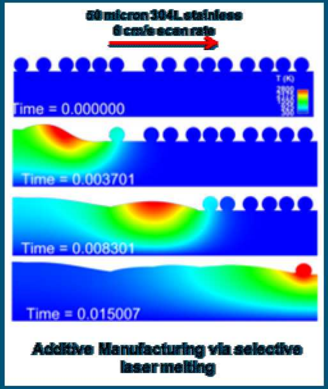
What is Aria?

Unstructured, nonlinear, multiphysics finite element solver

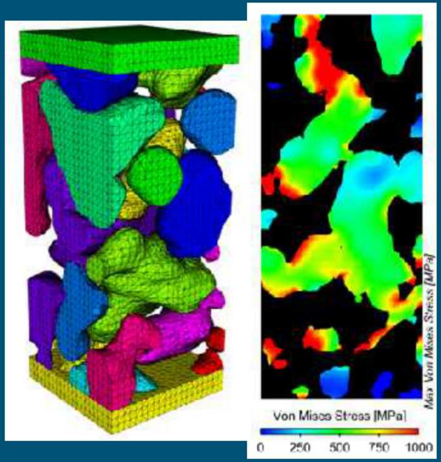
Implicit, full Jacobian

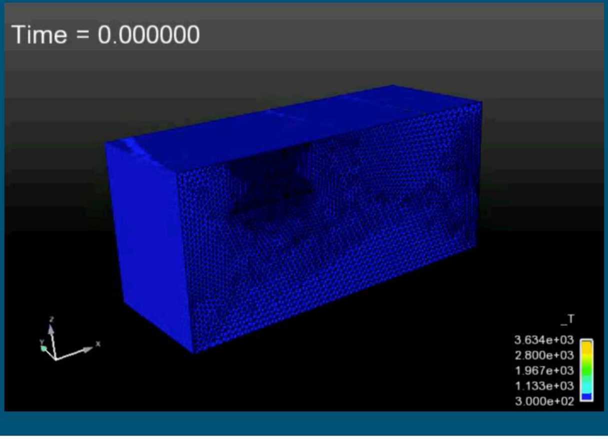
Key Libraries:

- Sierra Toolkit (STK)
- Trilinos linear solver stack

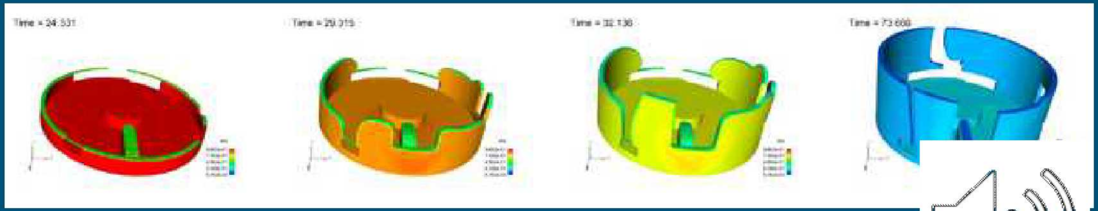


Additive Manufacturing via selective laser melting





COMPSIM
THERMAL FLUIDS



D. Noble, M. Martinez, R. Rao, S. Roberts, H

History of Aria Performance Portability

A Trip Back In Time

2001

November 2001:

First commits to the Aria codebase

Pentium 4, 1 core @ 2 GHz

My first high school CS class using Visual Basic

A Trip Back In Time

October 2012:

Titan is #1 on the Top500 as a hybrid CPU/GPU machine

Aria has made it 11 years as a CPU MPI-only code

A Trip Back In Time

October 2015:

First prototyping of threaded matrix assembly using Kokkos + STK in Nalu

- Co-design with Kokkos & Tpetra team members
- Drove creation of Kokkos scratch memory API

A Trip Back In Time

2001

2012

2015

2016

October 2016:

First prototyping in ariamini

- Started by pulling actual code from Aria
- Limited to just matrix assembly for steady state heat conduction
- Small enough amount of code to rapidly prototype, but always aware of how that will translate to the full application

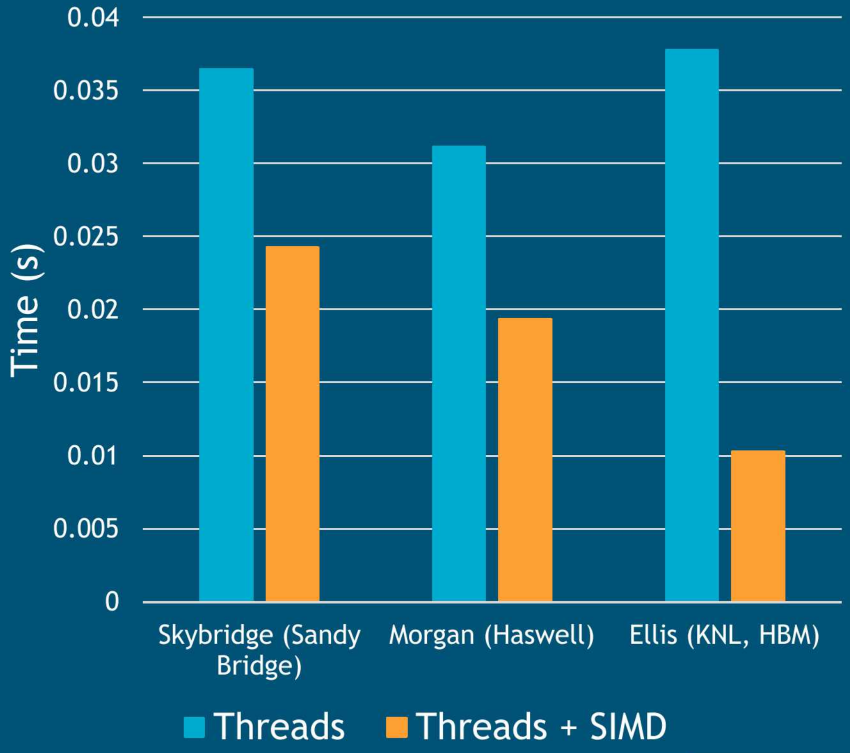
A Trip Back In Time

2001 2012 2015 2016 2017

February 2017:

Working performance portable matrix assembly in ariamini

- Use Kokkos::View inside main data structures
- Focused on OpenMP + SIMD for performance on Knight's Landing
- Functional on GPU, but no detailed performance exploration



A Trip Back In Time

2001

2012

2015

2016

2017

August 2017:

First step of Aria conversion based on ariamini

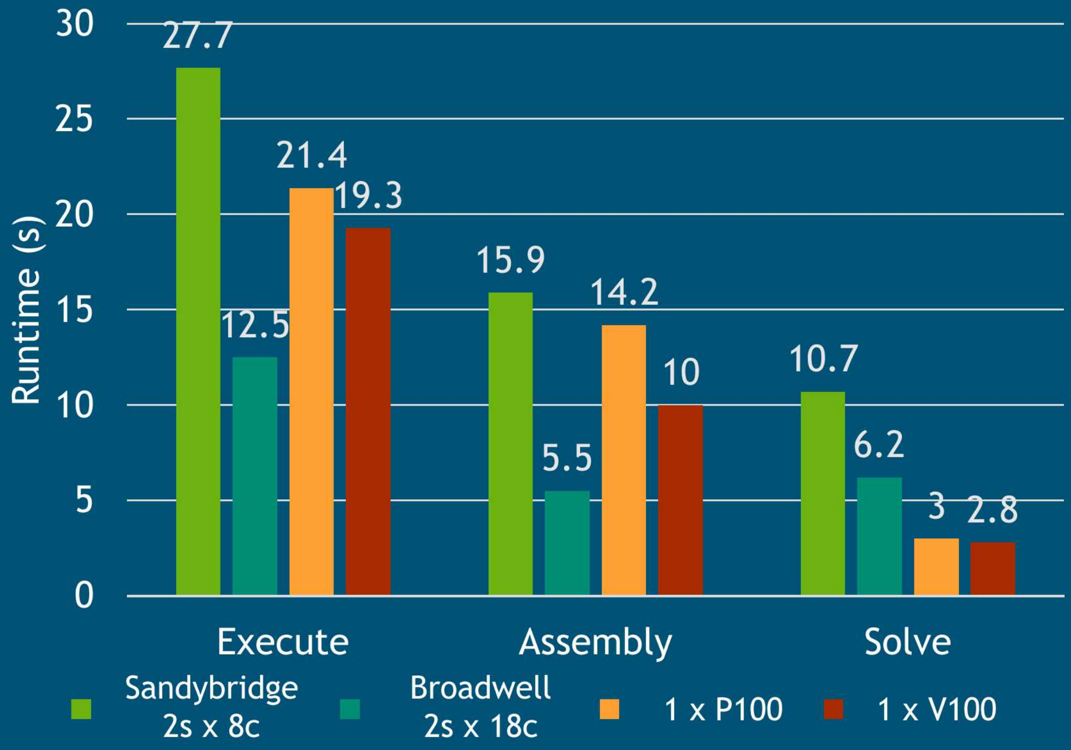
- Refactor whole Expression system to Kokkos-based data structures with SIMD support
- Interface to thread-parallel solvers based on Tpetra
- CPU threading only

A Trip Back In Time

2001 2012 2015 2016 2017 2018

December 2018:

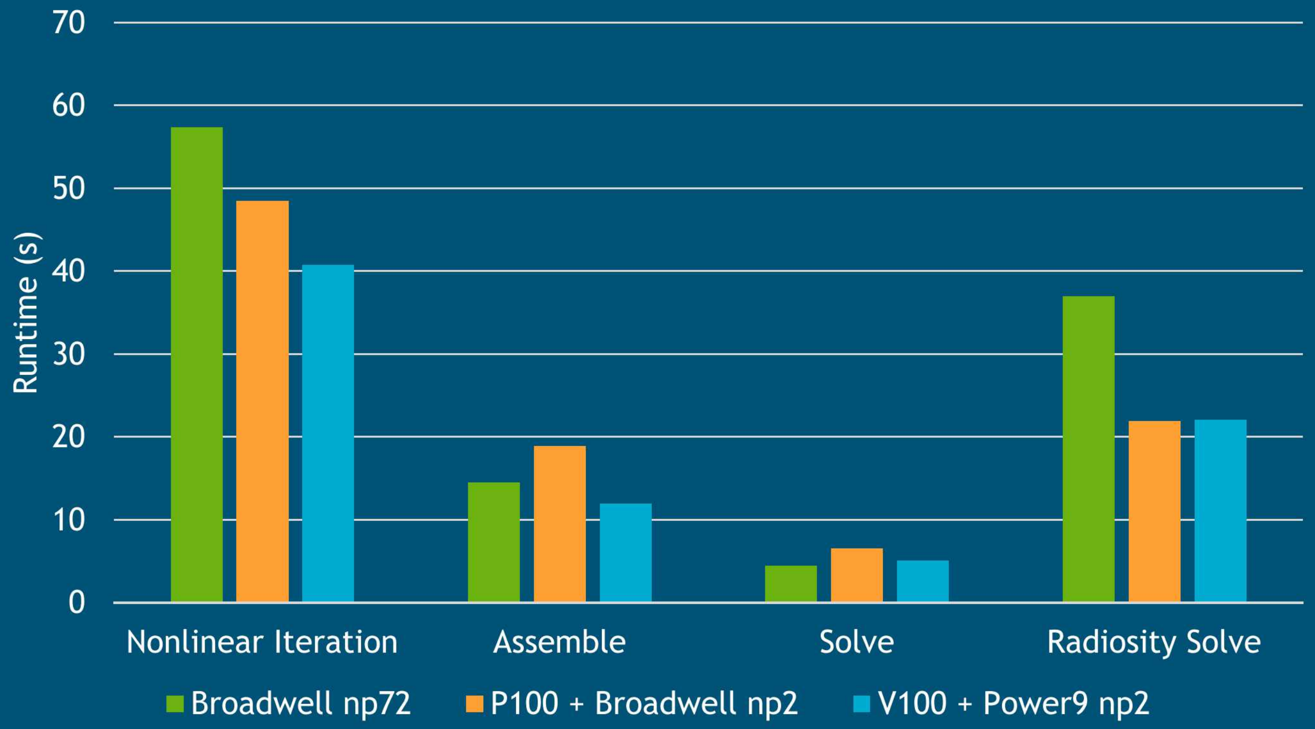
Initial GPU support in Aria for very basic conduction problems



A Trip Back In Time

August 2019:

Comparable performance between dual-socket Broadwell and GPU on realistic thermal problem

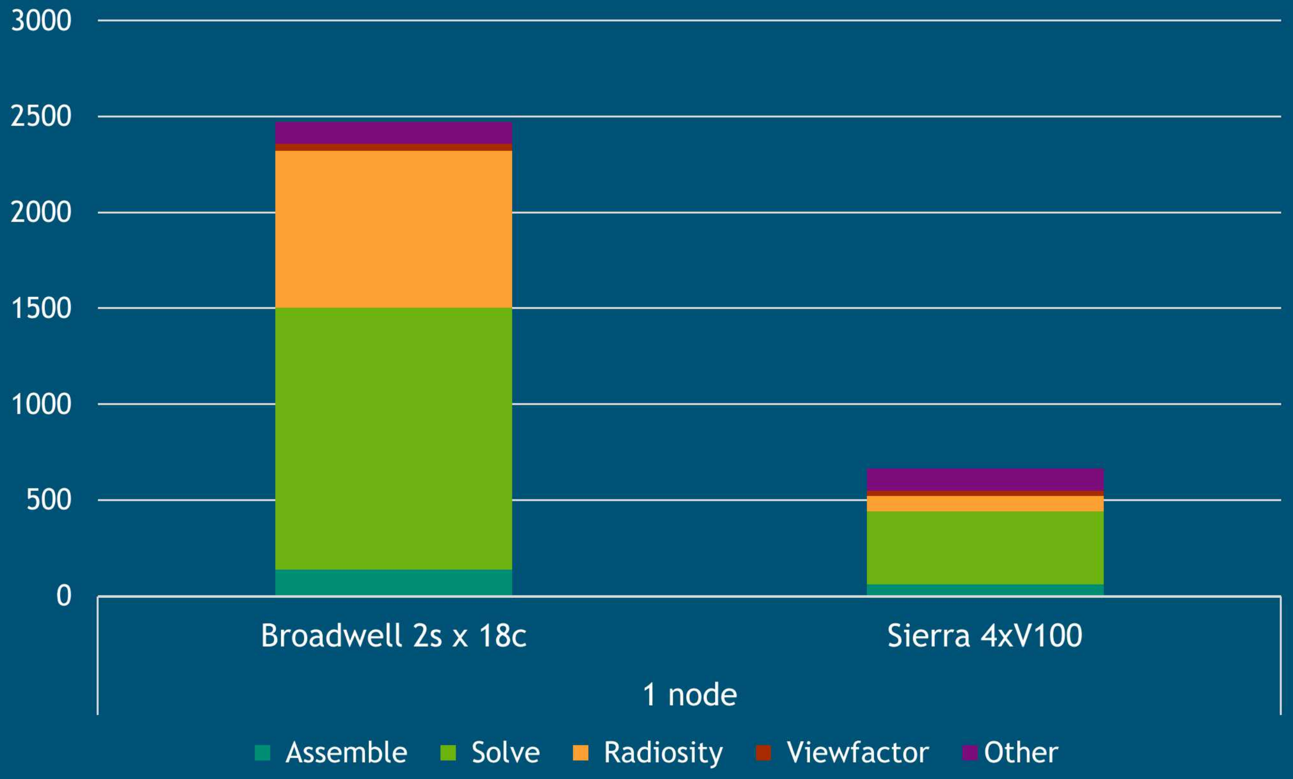


A Trip Back In Time

2001
2012
2015
2016
2017
2018
2019
2020

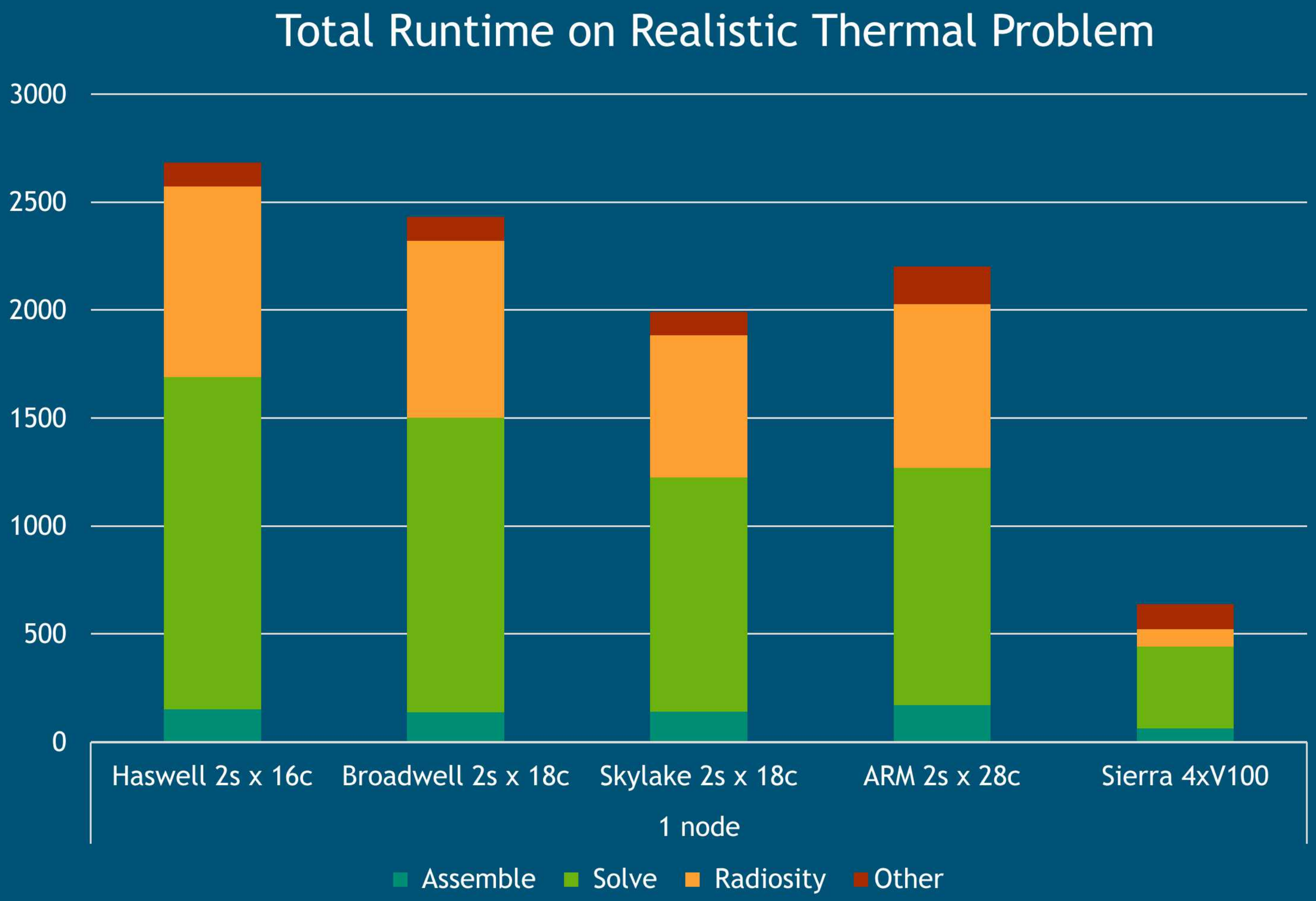
August 2020:

Sierra 3-4x faster than dual-socket Broadwell on realistic thermal problem



Current Performance Portability Results

Current Performance Portability



Lessons Learned

Working With Legacy Code is the Worst! … Right?

When does a reimplementation from scratch make sense?

- Is only a subset of the existing functionality needed (ever)?
- Is there no automated testing of the existing capability?
- Are you targeting an entirely new userbase?

I argue that if the answer to any of those is no, it is better to work with the existing codebase

- You may end up with a completely new implementation by the end

Make Legacy Code an Advantage

Existing test suite provides immense value

- Reproduces years of bugs
- Covers the unusual use cases users have that are easy to forget about

Extract key systems or kernels into miniapps

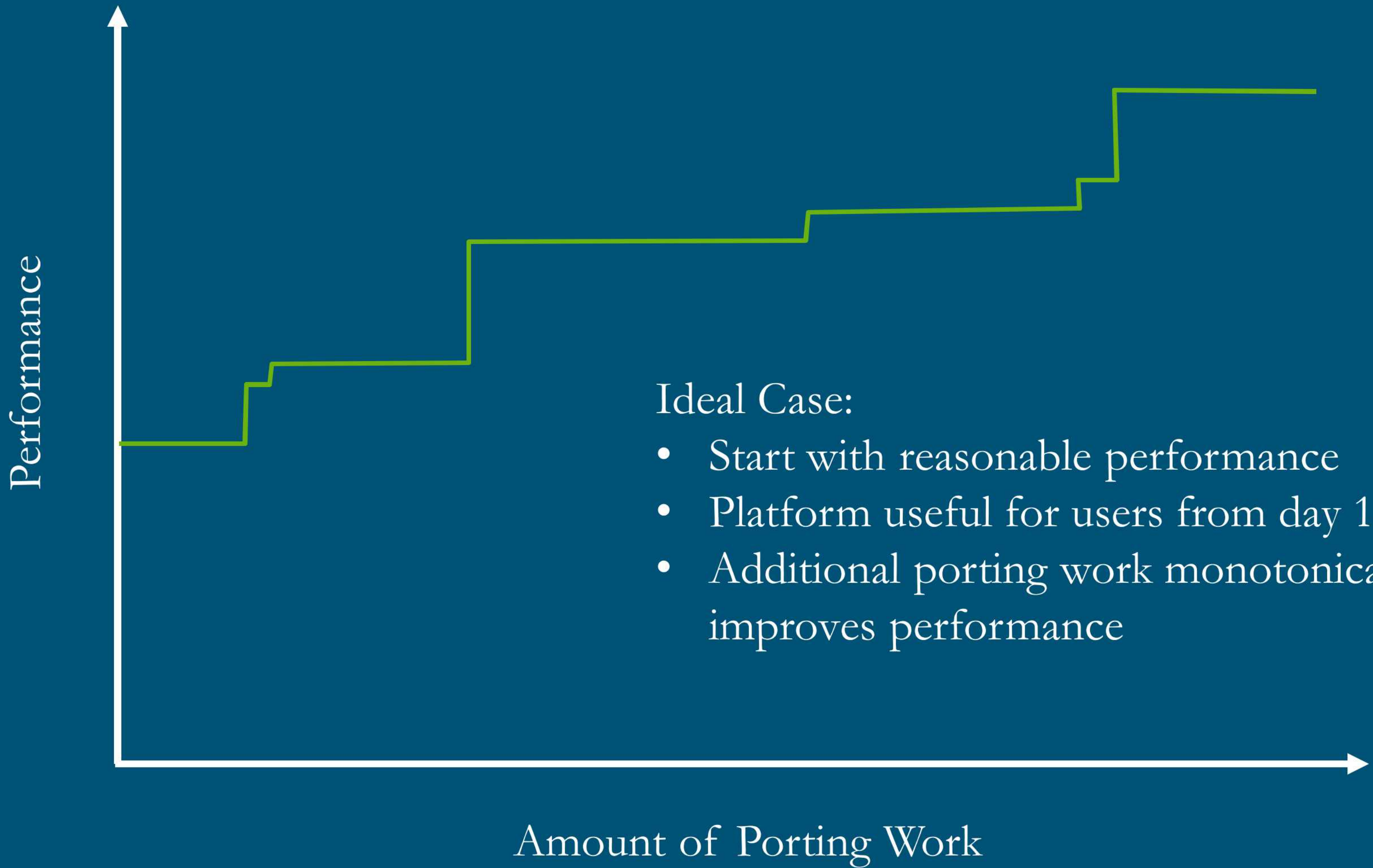
- Most of the prototyping flexibility you get from a reimplementation
- Easier to keep in mind the integration with the full application

Identify appropriate translation layers between new & old code as needed

Kokkos is a Starting Point for Application Performance Portability

- Basic building blocks for performance portability
  - Parallel loop patterns (for, reduce, scan)
  - Memory layout control (View)
  - Portable SIMD library (coming soon)
- Build application specific abstractions over Kokkos
  - Leverage application specific knowledge for performance & maintainability

The GPU Performance Portability Cliff



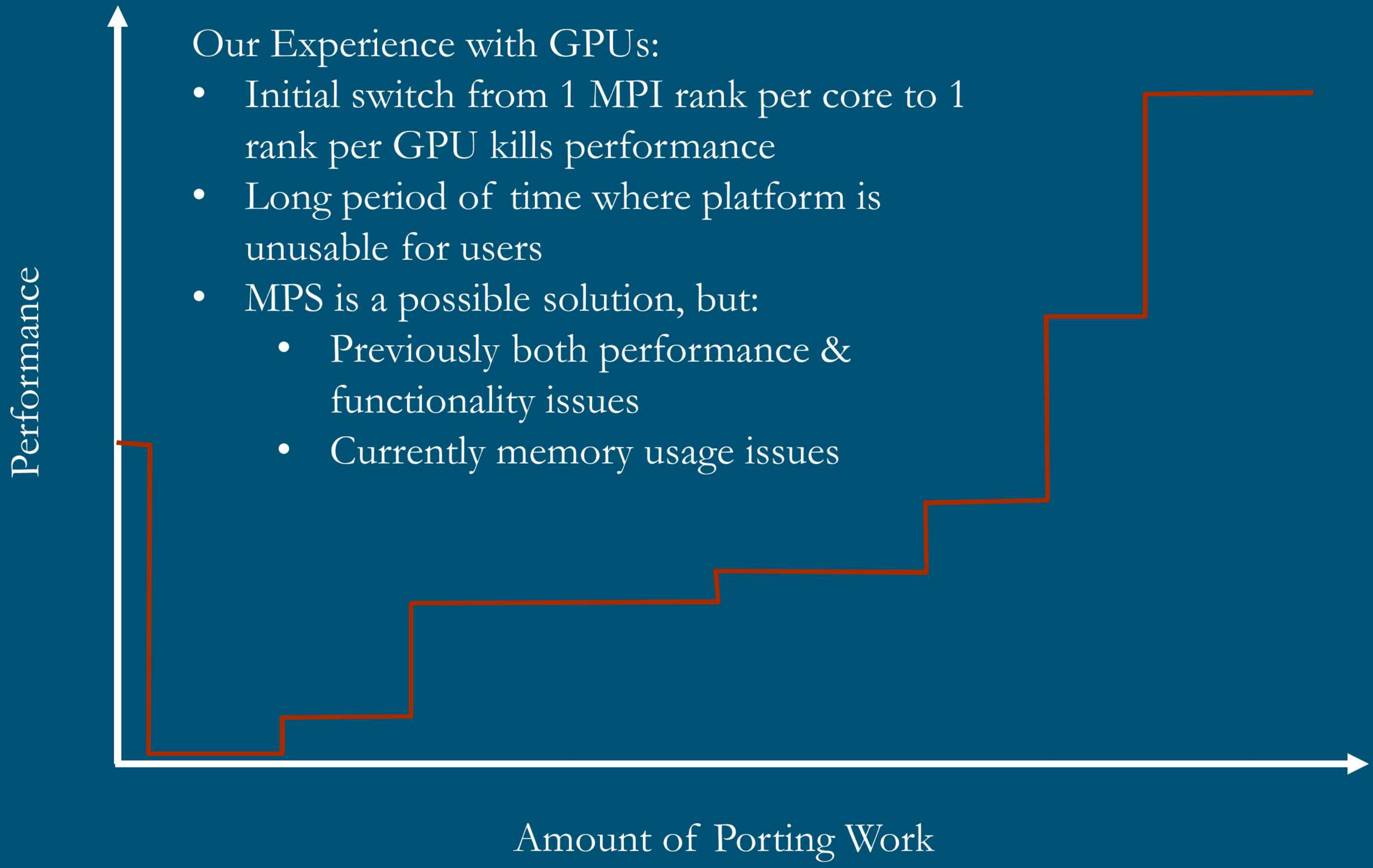
Ideal Case:

- Start with reasonable performance
- Platform useful for users from day 1
- Additional porting work monotonically improves performance

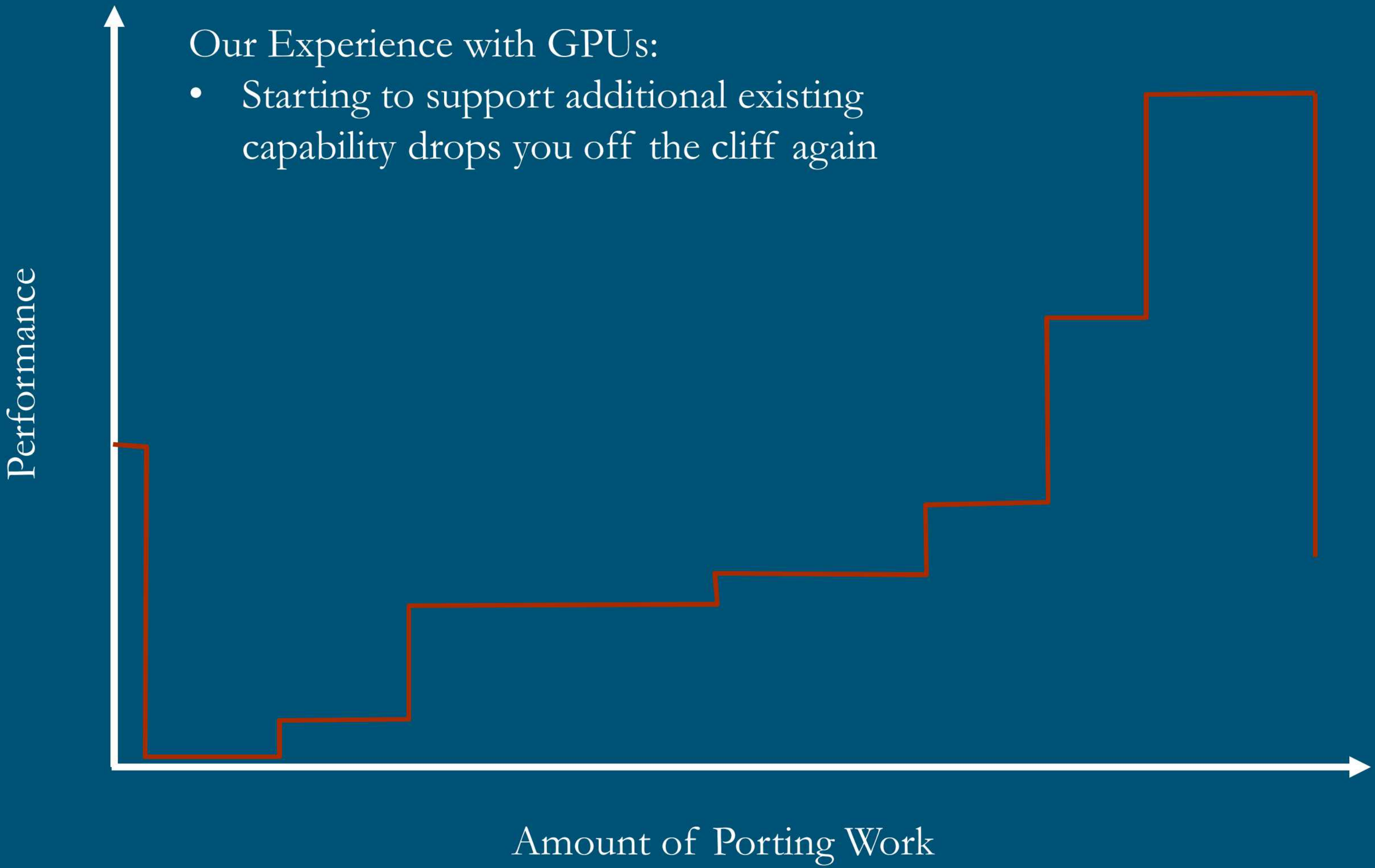
The GPU Performance Portability Cliff

Our Experience with GPUs:

- Initial switch from 1 MPI rank per core to 1 rank per GPU kills performance
- Long period of time where platform is unusable for users
- MPS is a possible solution, but:
  - Previously both performance & functionality issues
  - Currently memory usage issues



The GPU Performance Portability Cliff



Our Experience with GPUs:

- Starting to support additional existing capability drops you off the cliff again

Testing Challenges

Testing throughput on the GPU is a major issue

Aria has roughly 800 regression tests

- Vast majority are 1-4 MPI ranks and run in 1-10s on CPU platforms
- 3-5 minute runtime for total test suite with distributed testing
- Minimum 15s runtime in GPU builds
- Sharing GPU between multiple tests causes random failures
- > 1 hour runtime for total test suite in GPU builds

Questions?