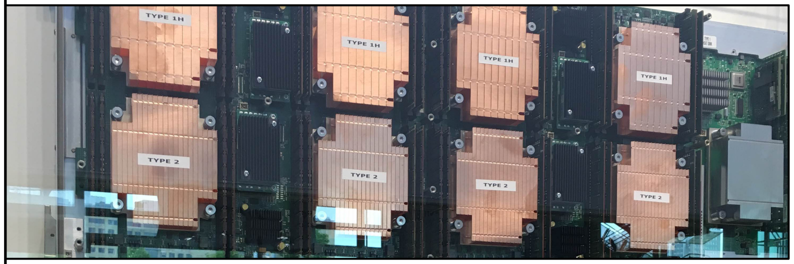
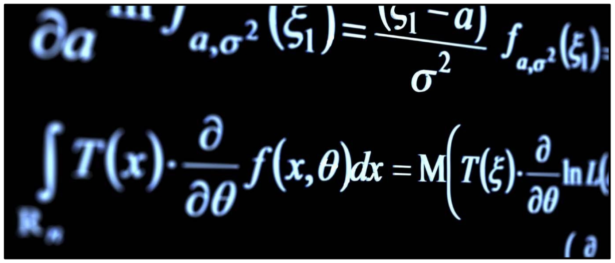
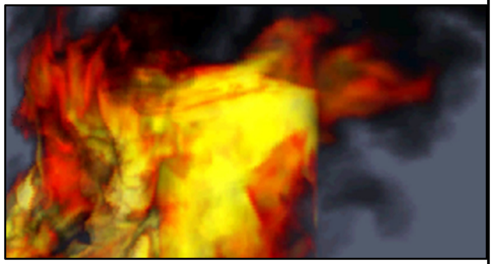
SAND2016-8523C

Exceptional service in the national interest







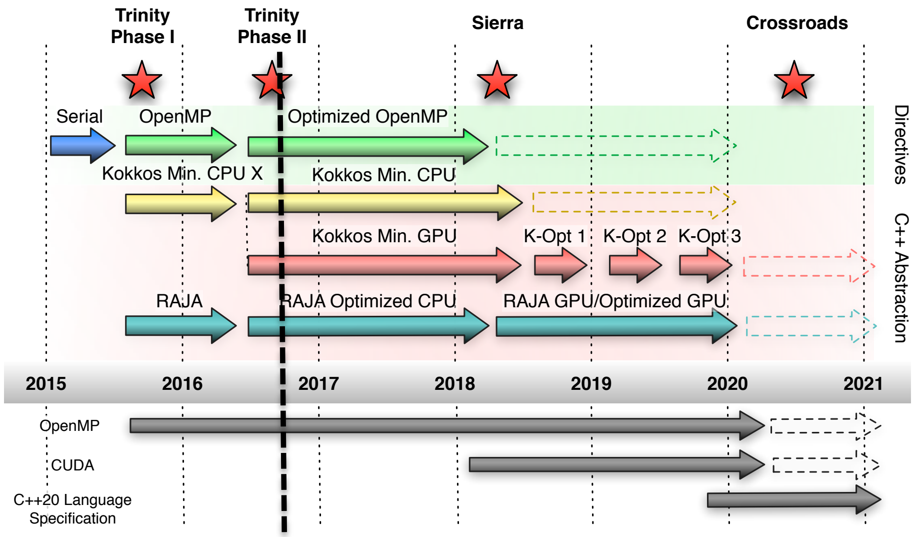
ASC Codesign L2 Milestone 2016

Center for Computing Research
Sandia National Laboratories/NM

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

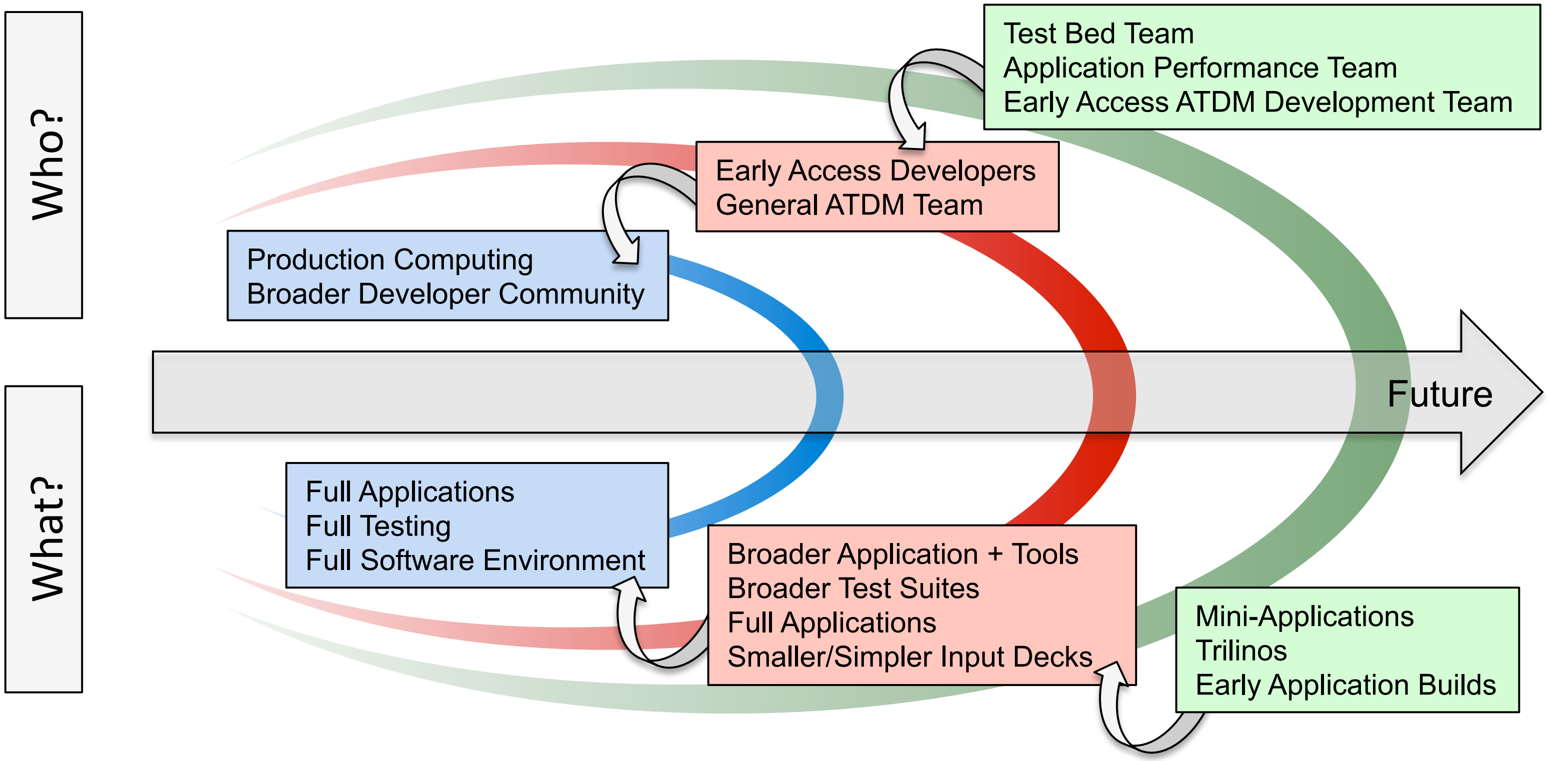
Foreword

- **ASC Codesign 2015 Plan**
  - Laid out a rough plan to get us to 2018 with minimal Kokkos
  - Optimization and refinement 2018-2020
- Significant progress in some codes
- SIERRA/STK now engaging for data structure design (for GPUs)
- Porting efforts now well underway for early cross platform results



Sandia ASC Codesign 2015 Proposal

Strategy for Next-Gen. Platforms



Test Bed Team
Application Performance Team
Early Access ATDM Development Team

Early Access Developers
General ATDM Team

Production Computing
Broader Developer Community

Full Applications
Full Testing
Full Software Environment

Broader Application + Tools
Broader Test Suites
Full Applications
Smaller/Simpler Input Decks

Mini-Applications
Trilinos
Early Application Builds

The Codesign Pieces

- In this presentation we are going to cover using our mini-apps, proto-apps and libraries to guide efforts in our:
  - Software Environment
  - Libraries and Tools
  - Applications
  - Hardware/Vendor Feedback

- This is a bit of a smorgasbord
  - .. but that shows the diversity of our work

Strategy for NGP Applications

- **Bring as much support as we can through our application frameworks (SIERRA, STK, Trilinos *etc.*)**
  - Reuse and share across our application portfolio
  - Single area can be worked on by domain/performance specialists

- Develop reusable recipes across our platforms with "known good" programming environments for prototype NGP systems
  - Seen significant improvement in developer productivity
  - Lowers total cost across machines
  - Faster time to deploy and get initial performance/testing etc
  - Important for packages which use CUDA, OpenMP etc. that are not in traditional environments yet

Outline

Sandia National Laboratories

- Its been a *very* busy year and we have a *lot* to show you…
- **Part 1 – Update on Application and Next-Generation Platform Porting (20 mins)**
  - Efforts to bring early environments on Haswell, KNL and POWER8/GPU
  - Early porting for SIERRA, RAMSES and Trilinos code projects
- **Part 2 – Performance Analysis tools and studies (10 mins)**
  - Focus on SIERRA/SD performance investigations
- **Part 3 – Experiences bringing Kokkos to application portfolio (30 mins)**
  - Bringing lessons/experiences from our mini-applications
  - Proto-applications (single physics) experiences
  - Connection to ATDM and IC

Part 1 – Application Porting to Next Generation Platforms

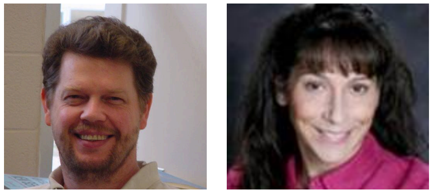
Bringing Haswell, KNL and POWER8/GPUs to our code teams

Application Performance Team

XL Compiler makes Paul grumpy

SIERRA Application Porting and Analysis

RAMSES Application Porting and Analysis



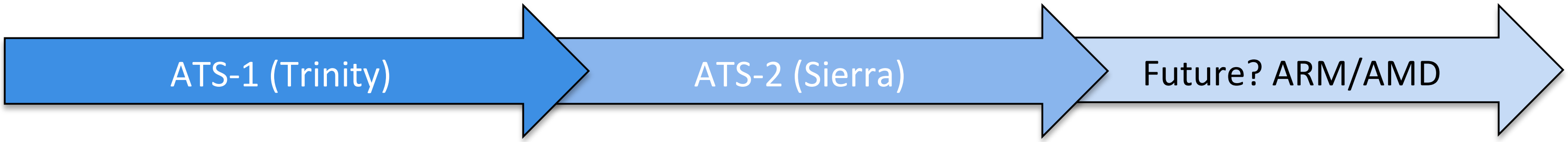
(Jeanine Cook and Doug Pase)

Performance Analysis Tools

Trilinos, Kokkos and Platform Tuning

+ Lots of Applications Teams and Support Input

Systems and Test Beds



ATS-1 (Trinity)

ATS-2 (Sierra)

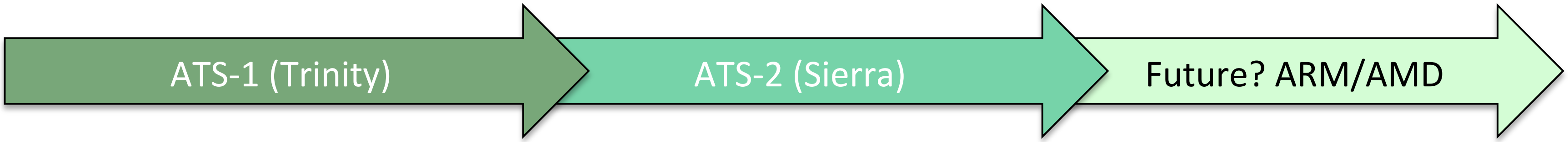
Future? ARM/AMD

- **Haswell**
  - Mutrino/Trinitite
  - Trinity
  - Shepard Test Bed
- **Knights Landing**
  - Mutrino/Trinitie
  - Trinity
  - Trinity white boxes
  - Ellis/Bowman Test Beds

- **POWER8/NVIDIA**
  - Ride/White Test Beds
  - K40/K80
- **General NVIDIA Test Beds**
  - Shannon
  - Shiller
  - Morgan
  - Hansen

- **ARM**
  - Sullivan Test Bed
- **AMD APU**
  - Cooper Test Bed

Software Environments



ATS-1 (Trinity)

- **Intel 15**
  - Generally production for SIERRA
- **Intel 16/17**
  - In Preparation
- **Cray**
  - Not seriously used
- **GNU**

ATS-2 (Sierra)

- **GNU 4.9 Series**
- **NVCC 7.5 and 8.0-RC**
  - Still remains buggy
- **GNU 5.3**
- **IBM XL 13.1.4/5**
- **PGI**

Future? ARM/AMD

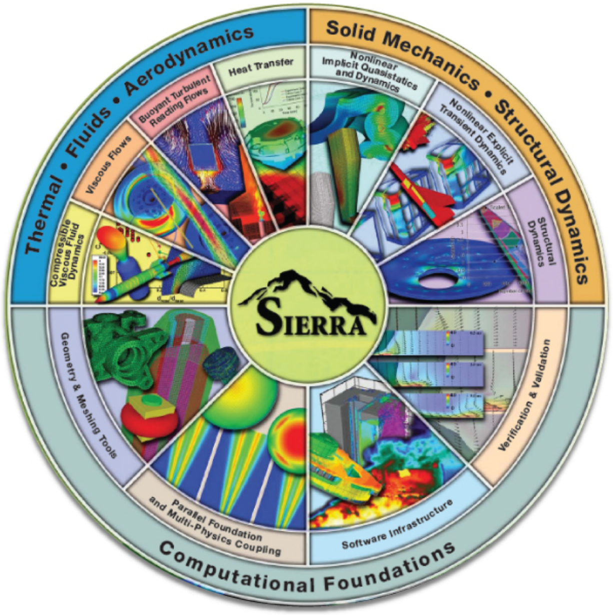
- **GNU 5.3 Series**
- **GNU 6.1 Series**
- **LLVM**
- **PathScale**

Production Applications Ported

Sandia National Laboratories

- **SIERRA**
  - Aero
  - Aria/MiniAria
  - Adagio
  - Salinas
  - Lots of others…
- **CTH**
  - Using OpenMP
- **CoE**
  - NALU/NALU-Kokkos

- **RAMSES**
  - Xyce (Serial & Parallel)
  - Charon2
  - ITS (using OpenMP)
  - SCEPTRE
  - And others…
- **ATDM**
  - ATDM-SPARC
  - EMPIRE


Thermal • Fluids • Aerodynamics
Solid Mechanics • Structural Dynamics
Computational Foundations
Heat Transfer
Buoyant Turbulent Reacting Flows
Viscous Flows
Compressible Viscous Fluid Dynamics
Nonlinear Implicit Quasistatics and Dynamics
Nonlinear Explicit Transient Dynamics
Structural Dynamics
Verification & Validation
Software Infrastructure
Parallel Foundation and Multi-Physics Coupling
Geometry & Meshing Tools
SIERRA

Utilize parts of Trilinos / Utilize some parts of Kokkos / Utilize some parts of both

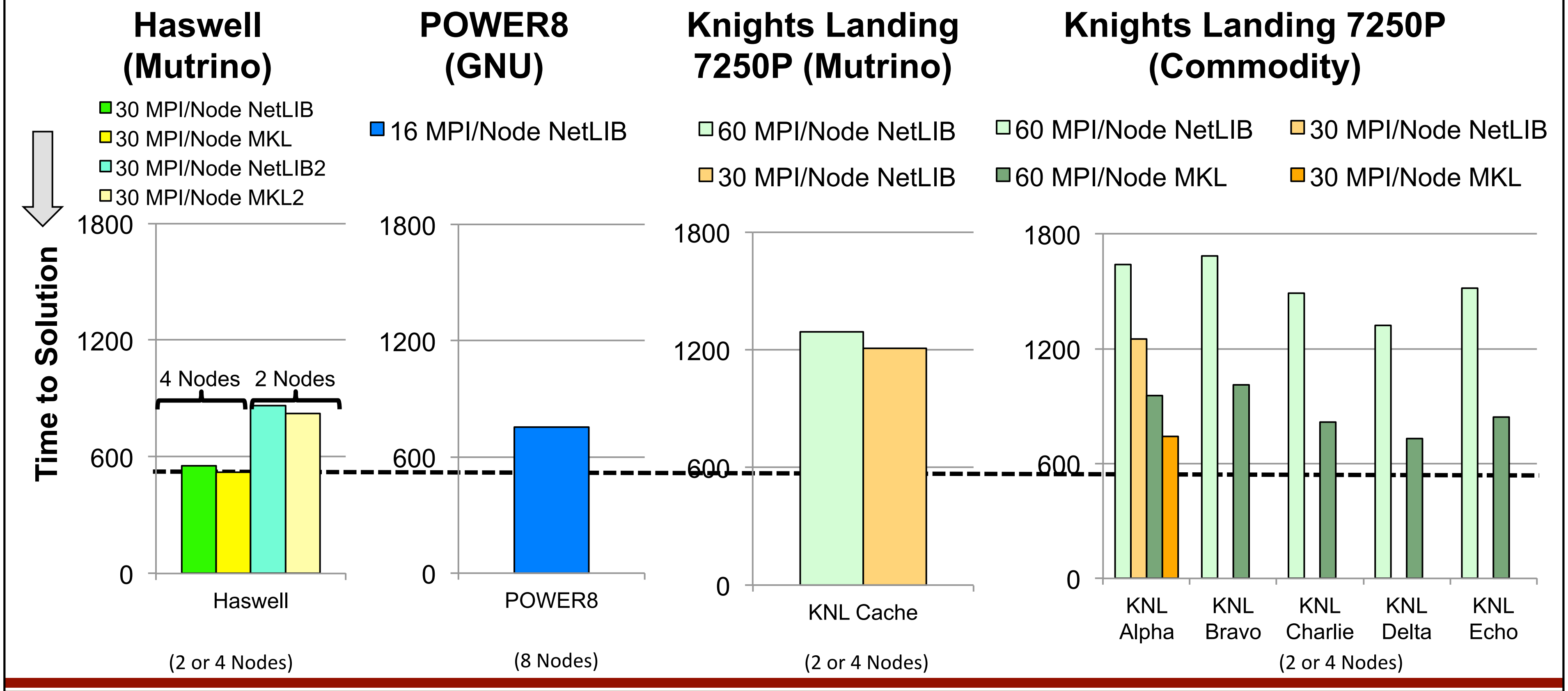
SIERRA General

- SIERRA Dev-Ops team and friendly users have been working on full suite compiles with NVIDIA CUDA compiler
  - Number of issues with C++ parsing etc.
  - Designed to support early development toolchain in preparation for ATS2
  - Significant progress

- Broadening SIERRA architecture support to include Cray-HSW, KNL and POWER8
- Very early builds with XL still showing number of issues which need to be worked on

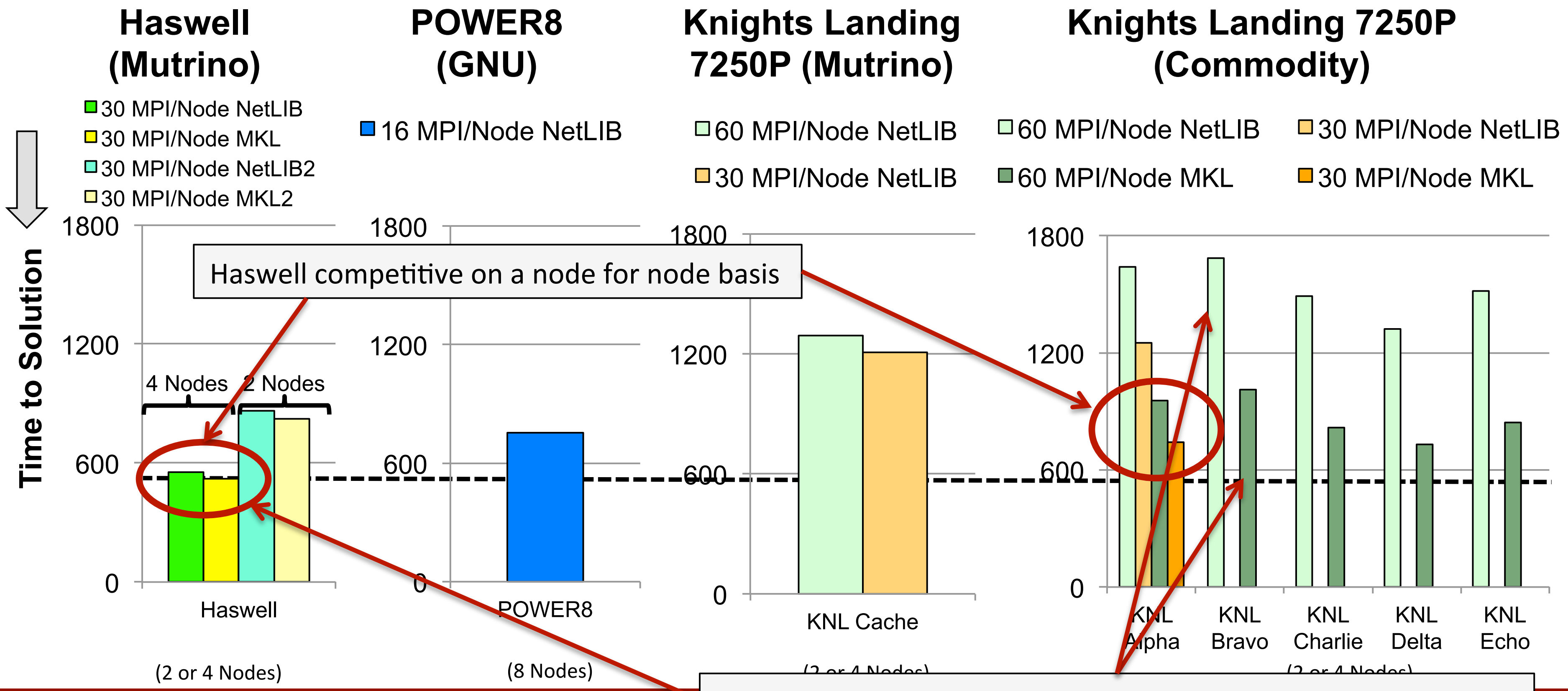
SIERRA/SD (“Salinas”)

Work by: Courtenay Vaughan and Paul Lin



SIERRA/SD (“Salinas”)

Work by: Courtenay Vaughan and Paul Lin



Haswell competitive on a node for node basis

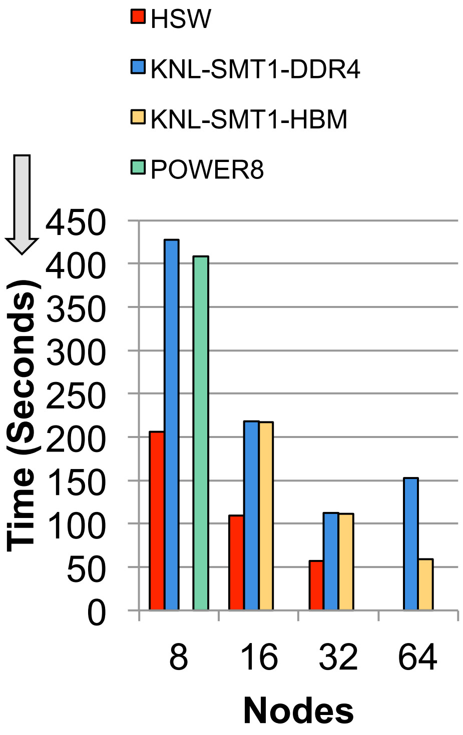
Much bigger difference in optimized libraries for KNL than HSW

SIERRA/TF (“Aero”)

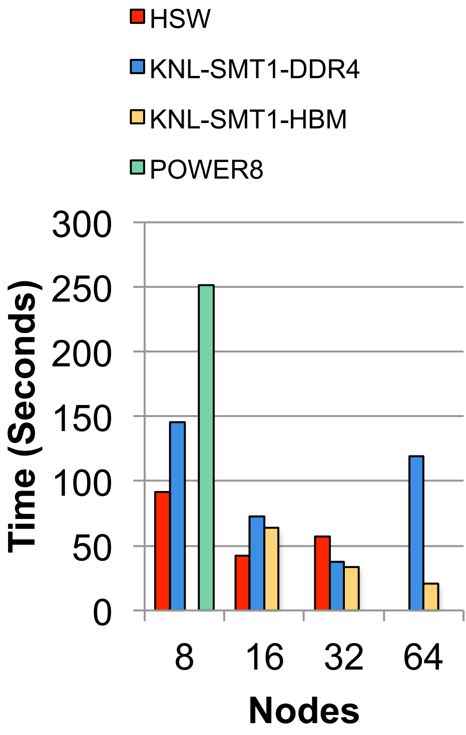
Work by: Paul Lin

Sandia National Laboratories

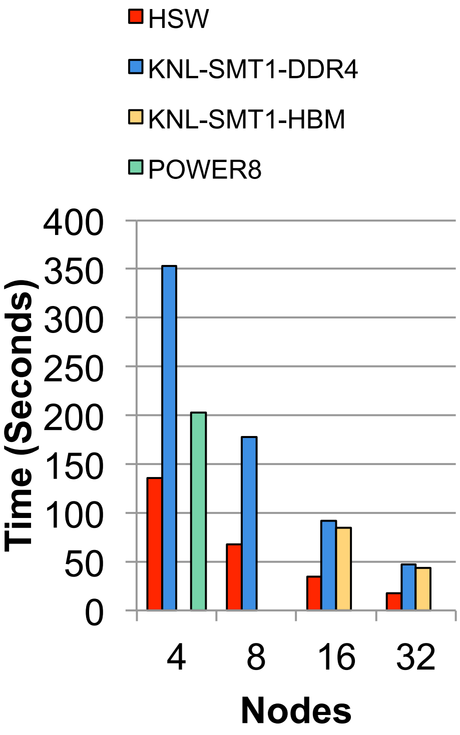
Matrix Assembly



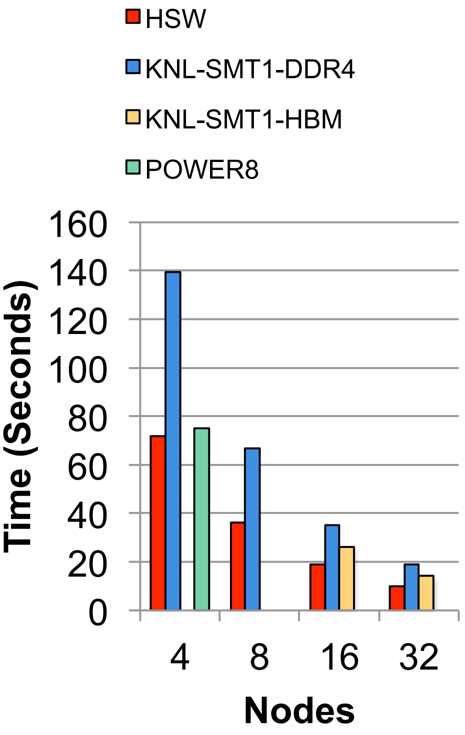
Matrix Solve



Matrix Assembly



Matrix Solve



Higher Order Problem

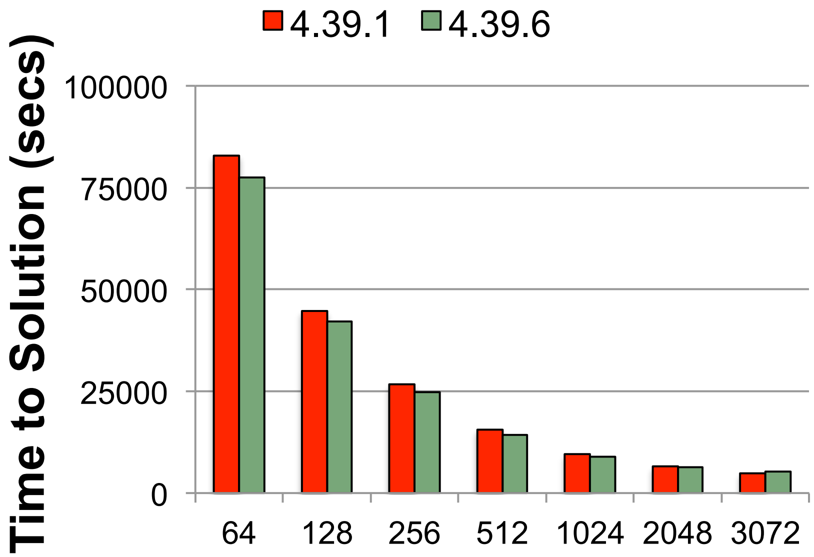
Standard Hex Mesh Problem

SIERRA/SM (“Adagio”)

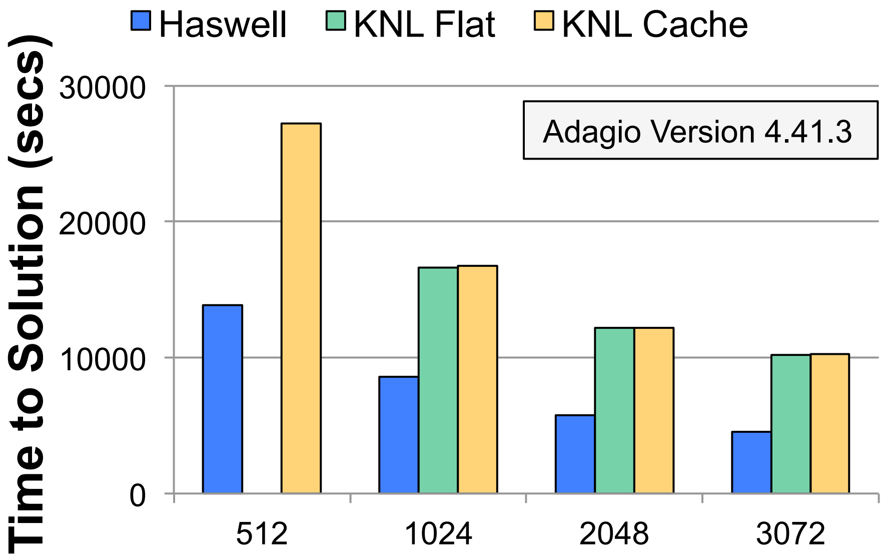
Work by: Dennis Dinge

Sandia National Laboratories

Code Improvement (HSW)



Trinity Preparation



Adagio Version 4.41.3

- See effect of continued improvement in code from SIERRA/SM team
- Greater use of well formed loops, const keyword, etc

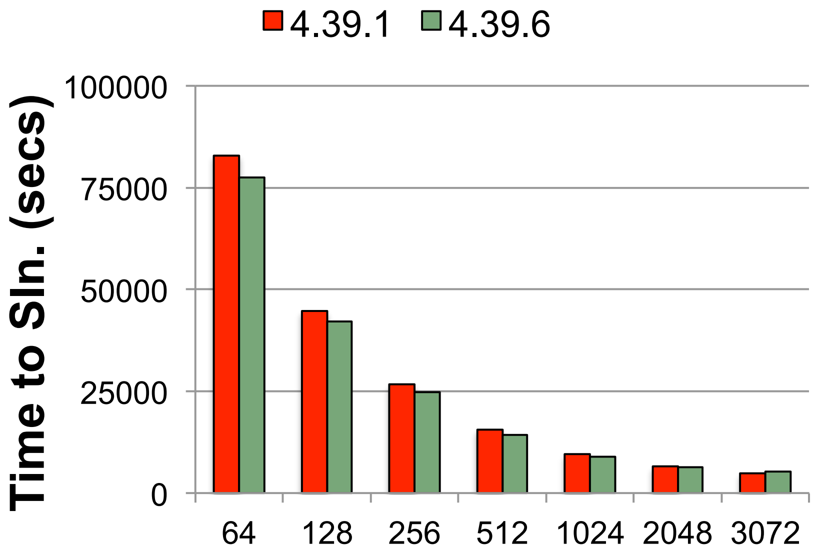
- Flat and Cache Mode are similar for KNL, problem does not fit at 512 ranks
- This is ranks, so compare HSW = 2 x KNL

SIERRA/SM (“Adagio”)

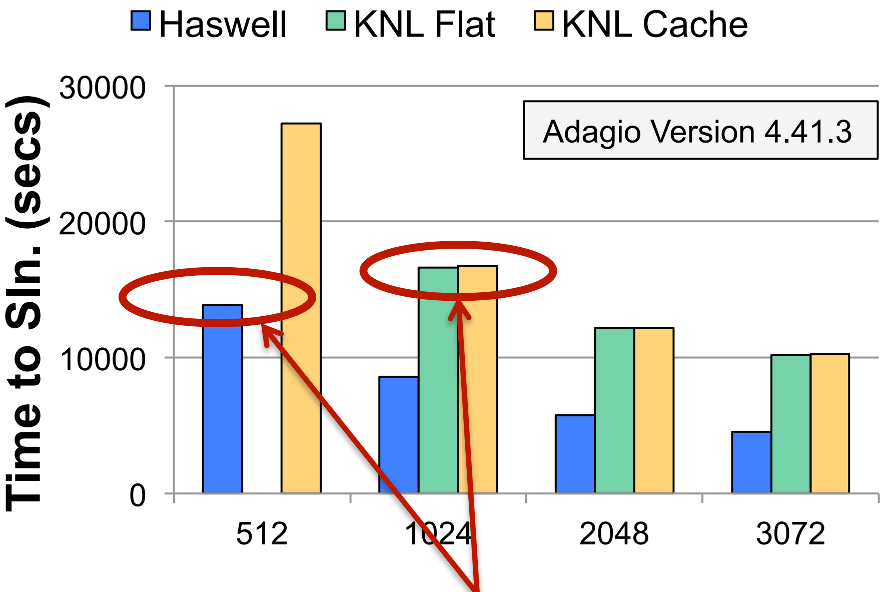
Work by: Dennis Dinge

Sandia National Laboratories

Code Improvement (HSW)



Trinity Preparation



- See effect of continued improvement in code from SIERRA/SM team
- Greater use of well formed loops, const keyword, etc

- Flat and Cache Mode are similar for KNL, problem does not fit at 512 ranks
- This is ranks, so compare HSW = 2 x KNL

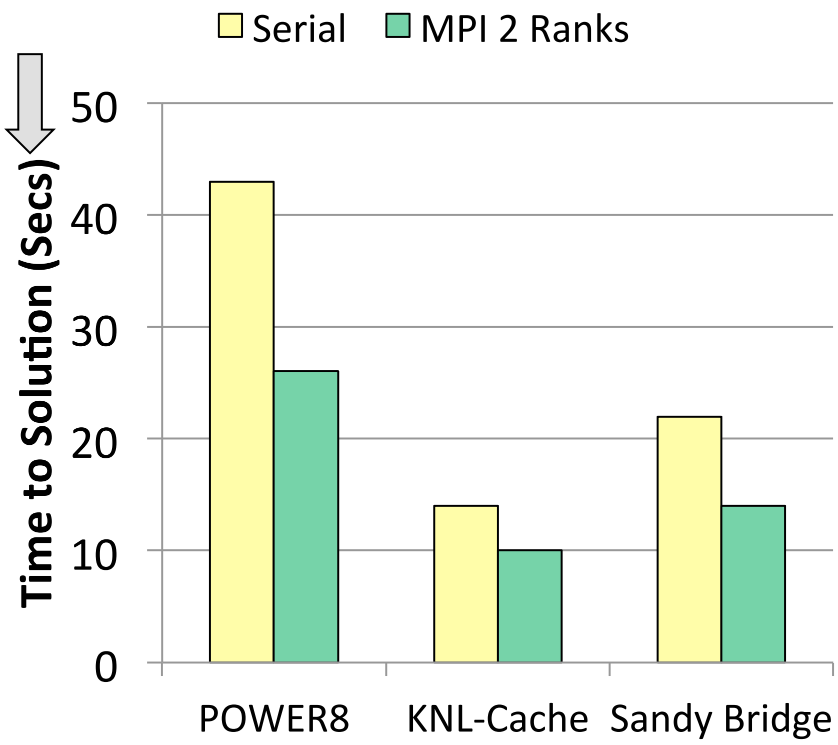
RAMSES - Xyce-Parallel

Work by: Bob Benner

- **Preliminary results from NGP platforms**

- Uses autoconf and so requires some significant cross-compile fixes
  - Continuing source of headache for **all** build systems we are using today (autotools, CMake and BJAM)
- KNL cache mode from Mutrino quite effective
- Working on >800 tests now

Xyce Parallel – BSIM6/RINGOSC17

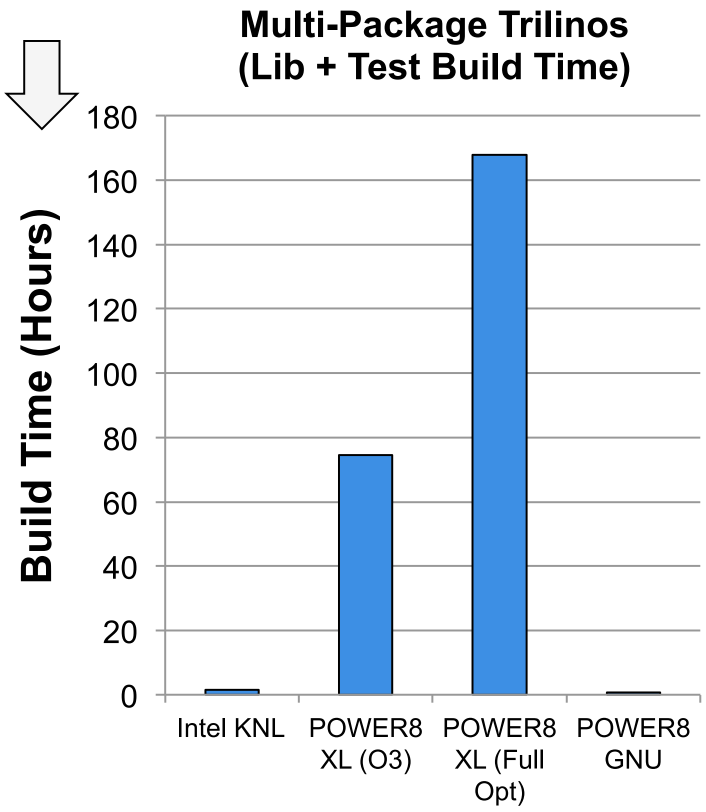


Performance Take Aways

- Seeing KNL is competitive (still bit slower) at comparable numbers of nodes
  - Surprising given the codes do not vectorize and only have very limited/zero threading today
  - Hopeful that continued work and development will get us to where we want to be
- KNL Flat/Cache modes are similar if you are all resident and can provide an improvement today (expect more in future with improved vectorization)
- Correlates with early experiences on test beds using mini-apps and Trilinos tests
- POWER8 bit slower than Haswell but this is using GNU compiler

Trilinos

- **Significant concern regarding compile times and binary sizes**
  - Trilinos testing infrastructure contains some miniapps (e.g. MiniFENL, Epetra-Driver *etc.*)
  - Measured compile and link times for large collection of packages
- Seeing this creep up significantly over time
- Particular concern at high optimization level with XL compiler
- Binary sizes for debug builds with Intel on KNL are worrying (sometimes seg-fault compiler)
- **Trilinos is now running on every NGP test bed**


Multi-Package Trilinos (Lib + Test Build Time)

Math Libraries Comparison

Library	X86	KNL	NV	P8	ARM	Model	Speed	BLAS	LAPACK	Sparse
NetLIB	🟩	🟩		🟩	🟩	Serial	🟥	🟩	🟩	
OpenBLAS	🟩	🟩		🟩	🟩	OpenMP	🟧🟨🟩	🟩	🟩	
MKL	🟩	🟩				OpenMP	🟩	🟩	🟩	🟨
CuBLAS			🟩			CUDA	🟩	🟩	🟨	
CuSPARSE			🟩			CUDA	🟩			🟨
ESSL			🟨	🟩		OpenMP	🟩	🟨	🟨	

High performing math libraries still critical to basic kernels, finding we don't always get broad support we need/want.

Takeaways

- **Efforts to port whole portfolio of production codes to NGP platforms now well underway for prototype/test bed ATS systems**
  - Significant angst with compilers (Intel 17.0 performance/bugs, XL compatibility and link times, Cray still not fully working)
  - Found *many* issues with environments (I/O libraries, compilers, configurations etc.)

- Good number of these are provided to vendors with fixes either delivered on in the works, some are on-going conversations
  - NVIDIA CUDA compiler has many problems in early access CUDA-8 RC1
  - Latest drops from PGI showing big improvement for threaded codes
- Math libraries continue to provide significant head aches cross machines
- Whole point is to grow broader lab NGP expertise which is underway

Part 2 – Performance Analysis

Improving our fundamental understanding of hardware performance

Background

- In the FY15 Tri-Lab codesign report we highlighted performance analysis tools on next-generation platforms as an area of real weakness
  - Takes a long time to get initial ports completed
  - Even longer to get the tool baked out for use
- Want something lightweight and simple to install across machines
  - Building on our multi-year investment in counter-based analysis
- Work has continued in this area and in building baseline capabilities
  - Shown in the context of the Salinas production code

PerfMiner - Overview

Lightweight profiling tools for our next-generation platforms

- Few dependencies
  - PAPI and Python
- Cross-platform
  - Defined default analysis so user doesn't have to
- Core, memory, MPI
  - Per rank, per thread, aggregated
- Modular design
  - Front-end, collector, viz back-end
- Viz back-end
  - Thorough "canned" analysis
  - Intuitive drill-down displays

- Directed analysis or continuous monitoring modes
- Data imported into searchable DB
  - Historical analysis
- Low overhead
  - Accurately multiplexes
- Configurable event list
- Flexible, simple instrumentation (PAPIEX_START()/PAPIEX_STOP())
  - Requires one re-compilation

Work by: Jeanine Cook and Phil Mucci

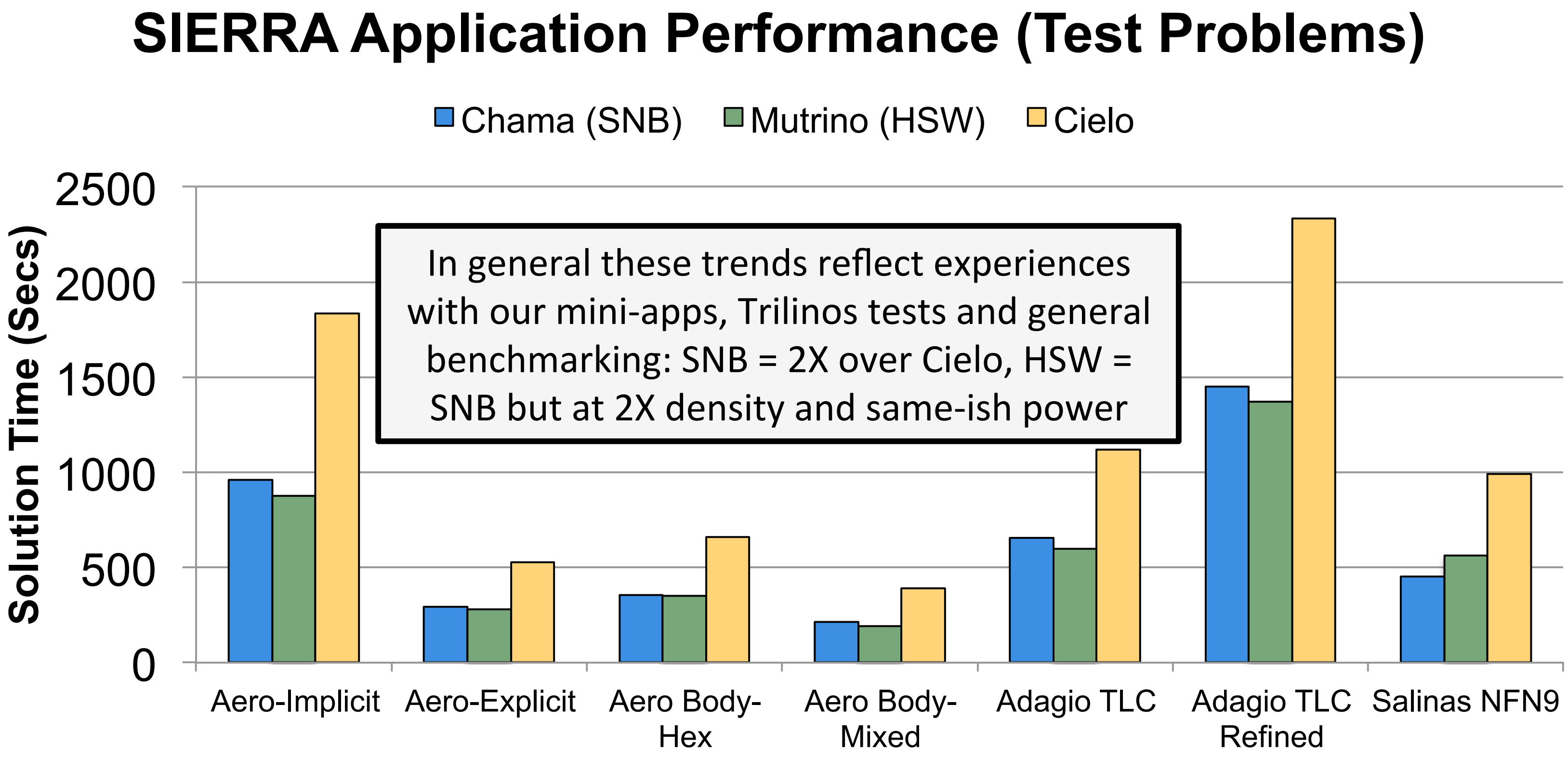
PerfMiner - Status

- Working and tested on:
  - Shepard, Chama, White, Ride, Mutrino test beds
  - (Haswell, Sandy Bridge, POWER8 and Haswell/Cray)

- Defined detailed analysis for HSW, SNB/IVB, POWER8
  - Cache behavior
    - Miss rates
    - Pending and stall cycle histograms
  - Memory latency and cycle accountability histograms
  - Execution port utilization
  - Micro-op execution distribution
  - Branch behavior
  - Memory bandwidth estimation
  - CPI (cycles per instruction)
  - Instruction retirement behavior
  - Resource stall profile (Future: CPI stacks)

SIERRA Performance

Sandia National Laboratories


SIERRA Application Performance (Test Problems)

In general these trends reflect experiences with our mini-apps, Trilinos tests and general benchmarking: SNB = 2X over Cielo, HSW = SNB but at 2X density and same-ish power

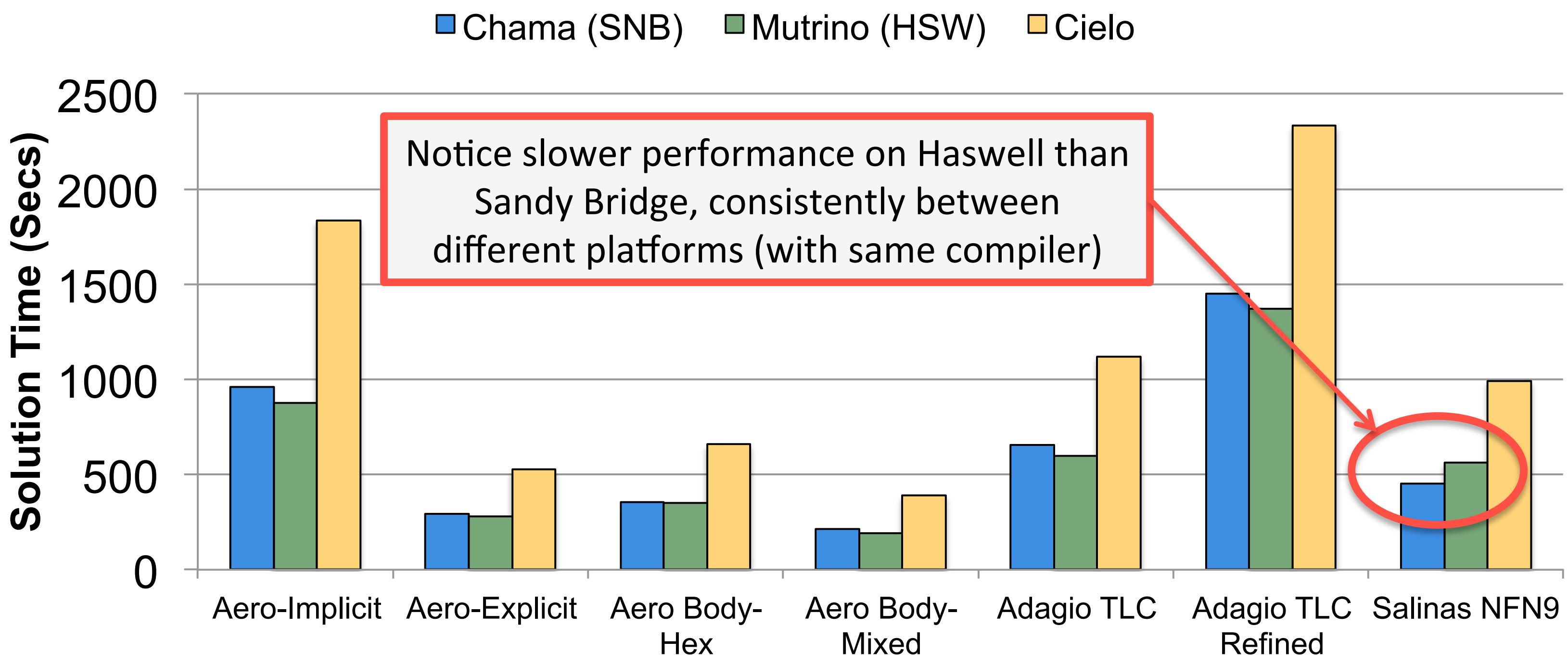
These are for a fixed number of MPI ranks (so are using different numbers of nodes)

Work by: Courtenay Vaughan

SIERRA Performance

Sandia National Laboratories

SIERRA Application Performance (Test Problems)



Notice slower performance on Haswell than Sandy Bridge, consistently between different platforms (with same compiler)

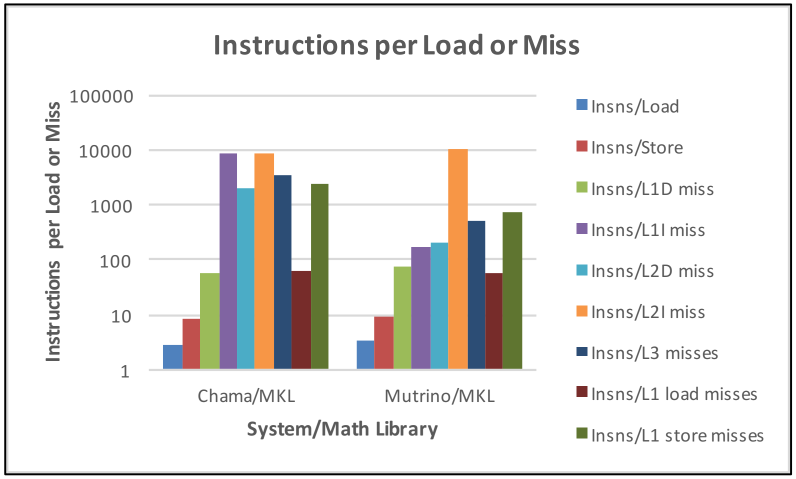
Analyzing Why Different?

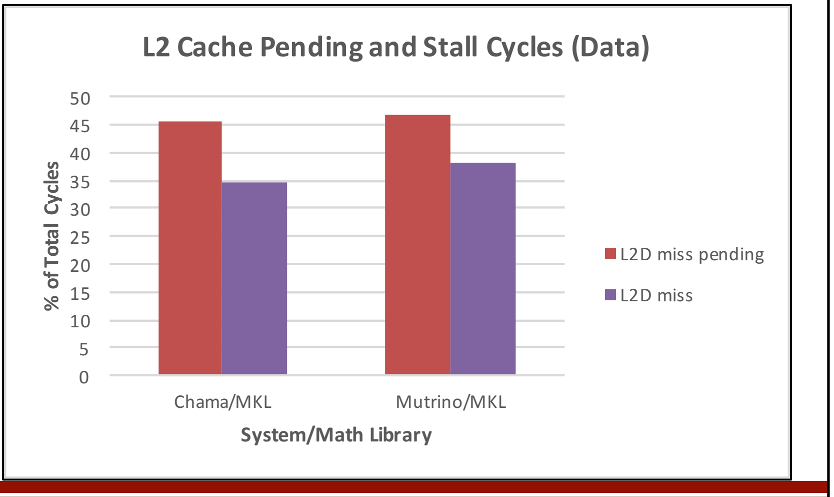
- Initial code-based times were showing that the Salinas BLKSLV routine was where additional cost was going
  - Frequent calls to small-ish DGEMM and/or DGEMV
  - Options to link to Cray LibSCI, Intel MKL and native NetLIB implementation
  - No real significant different between the libraries (small DGEMM continues to be a huge headache)

- Requires more in-depth analysis of the solve routines from within the application context
  - **Why?** – turns out we don't have a representative mini-app for this type of solve
  - Good motivation for a tool that can scale to more complex applications and multi-rank

BLKSLV Analysis

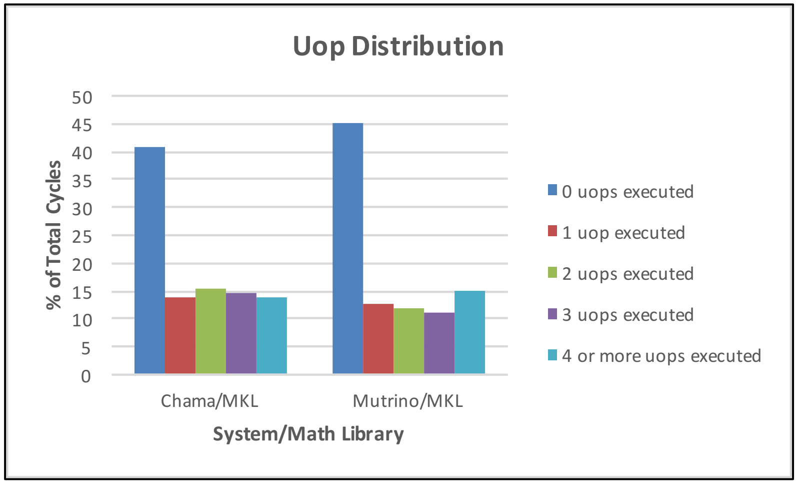
- Instructions per Load/Miss (higher is better)
  - Mutrino has comparatively poor Icache, L2, and L3 miss behavior
    - Enough to explain ~10% performance difference?
- L2 Cache Pending and Stall Cycles
  - Mutrino spending more time (but not much) on waiting for L2 pending misses and stalling more to resolve L2 misses
    - Enough to explain ~10% performance difference?

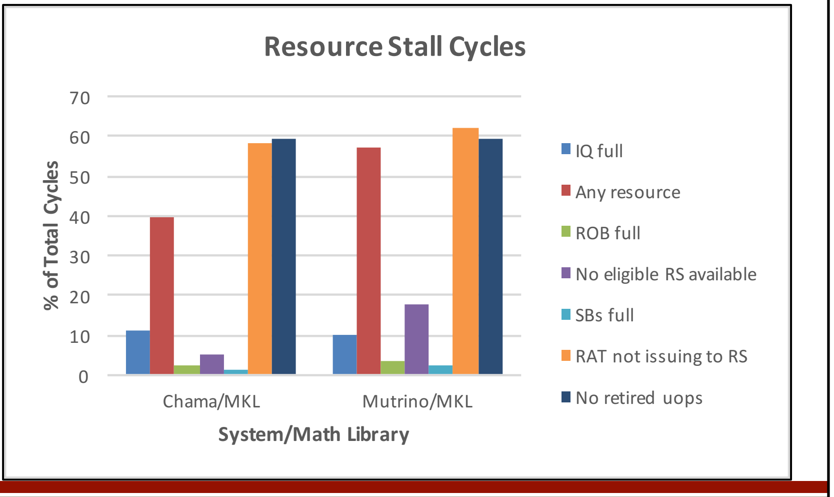




BLKSLV Analysis

- Uop distribution
  - Mutrino characterized comparatively by more cycles with fewer uops executing
    - Could this be an artifact of the cache behavior?
    - Enough to explain ~10% performance difference?
- Resource stall cycles
  - Mutrino spending comparatively more cycles stalled on any resource, on no reservation station available
    - Could this be explain the uop distribution?
    - Enough to explain ~10% performance difference?





Discussion

- PerfMiner provides capability to analyze execution relatively easily and thoroughly
  - Other tools really don't provide same flexibility or robustness
  - Reveals problems that can either be used to explain anomalies or to drive further architectural exploration (SST)
    - Identify the problem, but not the cause -> future work!
  - Issues with performance counters
    - Can you use them to compare cross-platform performance?
      - Simulation could help answer this
- Application performance understanding really just in beginning stages
  - PerfMiner capability is new
  - APT team process for doing this still being ironed out

Takeaways

- Frustrating situation is that we often don't have performance tools on our machines early in the cycle (VTune, virtually nothing on POWER8, *etc.*)
- Mini-application performance shows Haswell should out-perform Sandy Bridge which in general is correct except for Salinas
  - Almost all other applications show Haswell outperforming Sandy Bridge by around 2X
- Requires us to be able to go into the systems/applications and understand why to reduce codesign cycle feedback times
  - Important in very early system bring up where we can affect a lot of change
- Now baseline capabilities are developed working on how we impact application designs with continuous performance assessment capabilities (tasking into FY17)

PART 3 – APPLICATION DEVELOPMENT FOR NEXT-GENERATION PLATFORMS

Bringing experiences and best-practices to our application portfolio

Section Outline

- **Reminder – observations of performance from SIERRA/TF Aero**

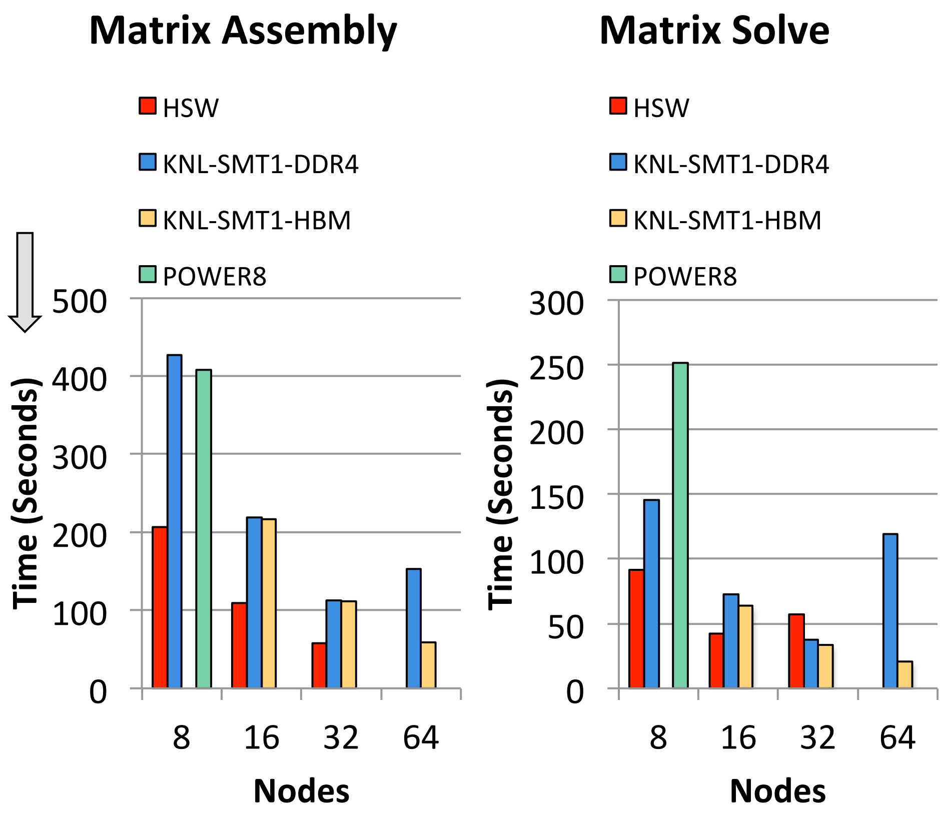
- What do we see in our mini-app and proto-app suite?
  - Similar trends? Similar performance?

- Some findings and optimizations worked on during this codesign cycle
- Implications for other codes in our portfolio
- Broader impact

SIERRA/Aero

Work by: Paul Lin

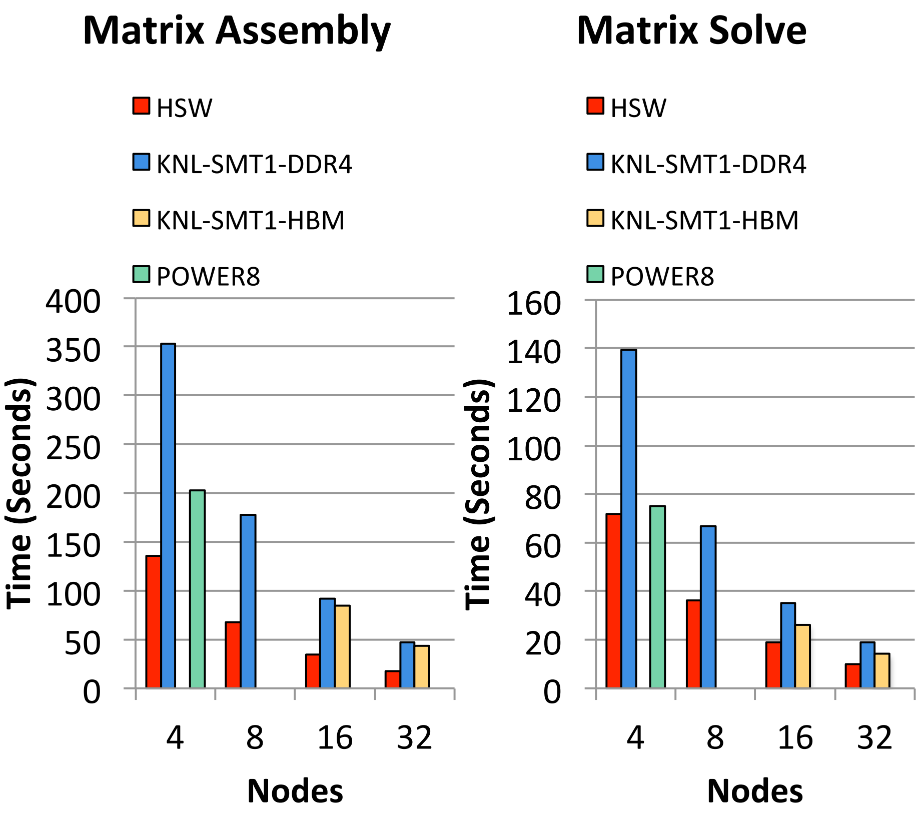
Sandia National Laboratories



Matrix Assembly

Matrix Solve

Higher Order Problem



Matrix Assembly

Matrix Solve

Standard Hex Mesh Problem

Observations

Assembly

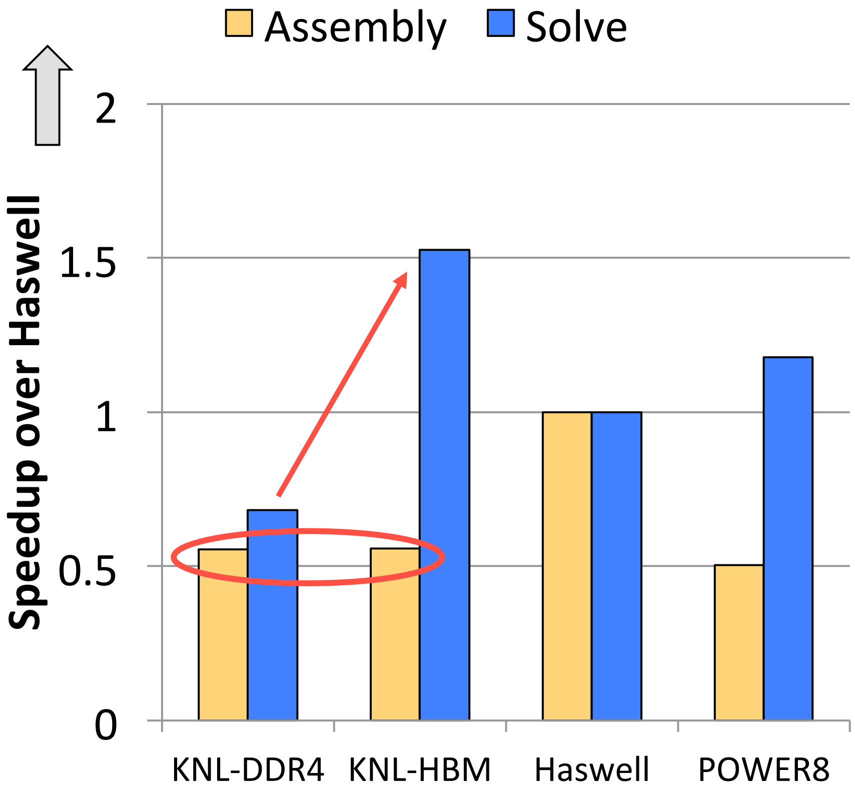
- Assembly is not improved in most cases by HBM on KNL
- Tends to be slower than Haswell for equivalent compute resources
- POWER8 worse than Haswell

Solve

- HBM is faster than DDR4 but not as much as we would like (solve is too simple)
- See better scaling when using HBM (at larger node counts)
- Seeing similar results with Trilinos testing and development

MiniFE

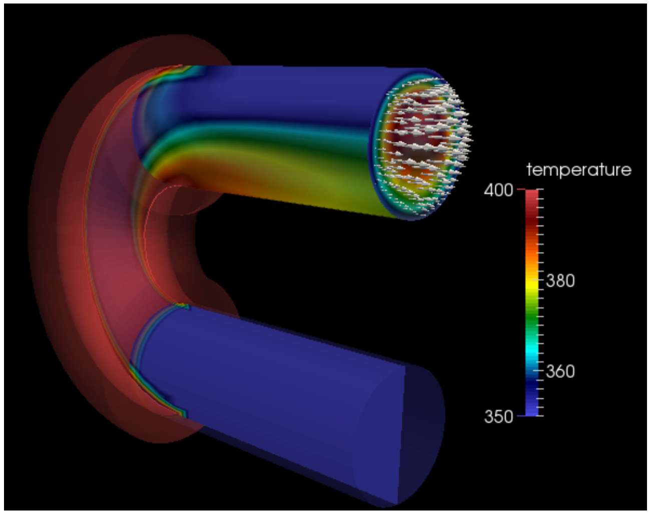
- Running in MPI only mode using **only cores**
- Compiled with optimized flags on each platform
- See little difference in Assembly times between DDR4 and HBM on KNL
- KNL is slower than Haswell
- Solve is faster on POWER8 and KNL-HBM where memory bandwidth is higher



See similar behavior in HPCG benchmark runs but solve is slowed down by serial GS

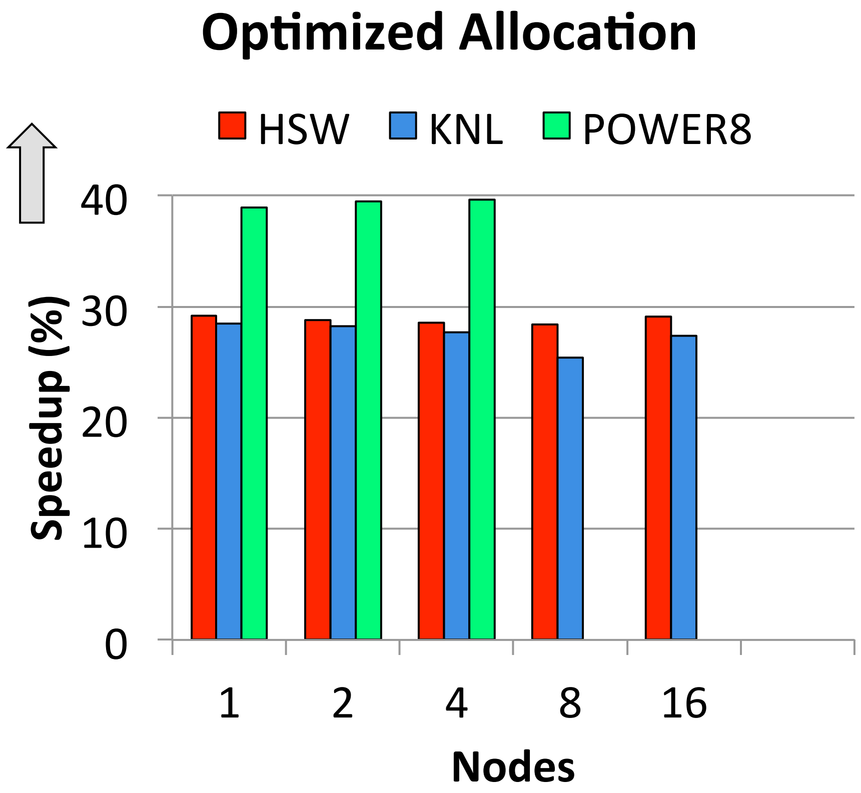
NALU Proto-App

- CFD code
- Real application, but simpler than full NW apps
  - Fewer dependencies
- Shares data structures, algorithmic approach and Trilinos usage with full NW apps
- Prototype Assembly Porting:
  - Use of Hierarchical Parallelism: Loop over Buckets and Elements in Buckets
  - Replace temporary allocations with Kokkos scratch memory



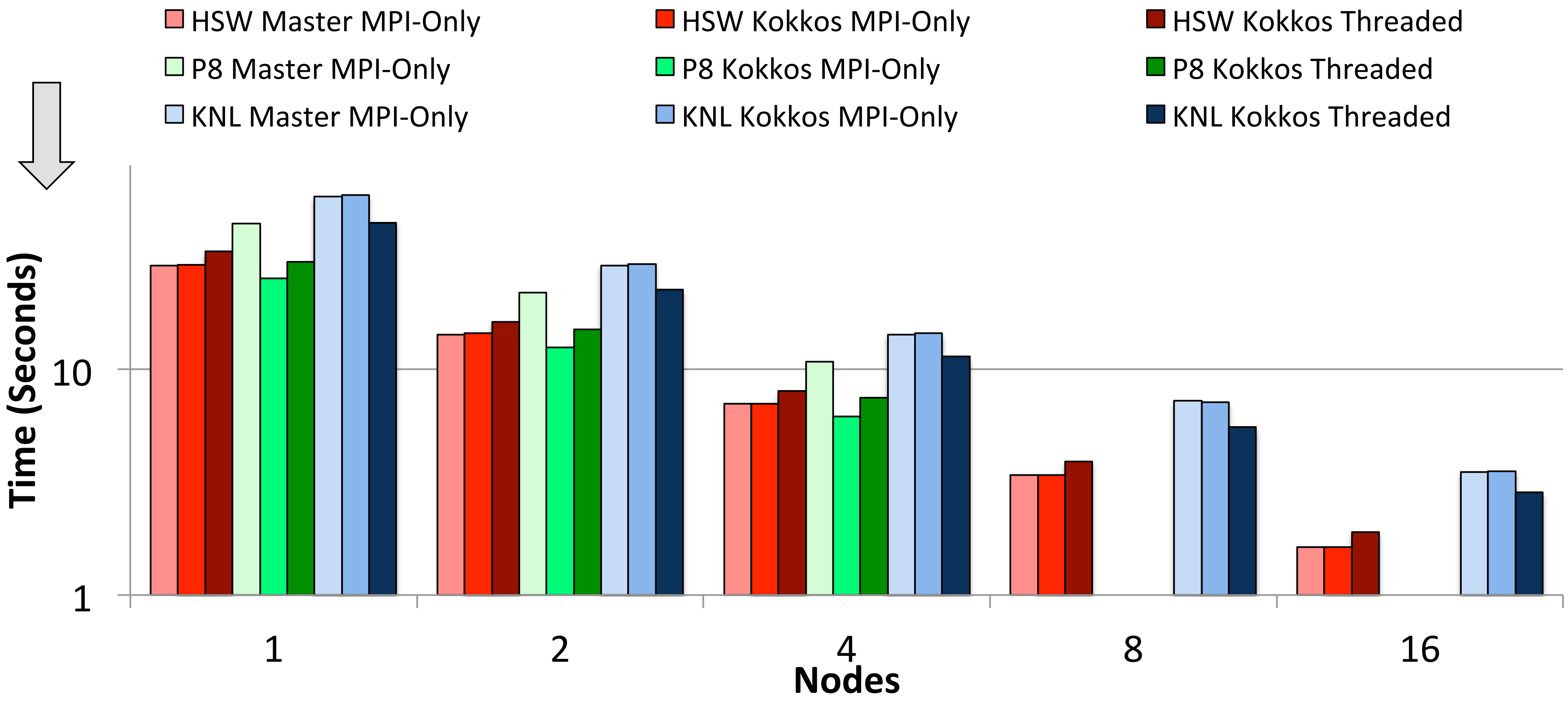
https://github.com/spdomin/Nalu

Optimizing Allocation in Assembly

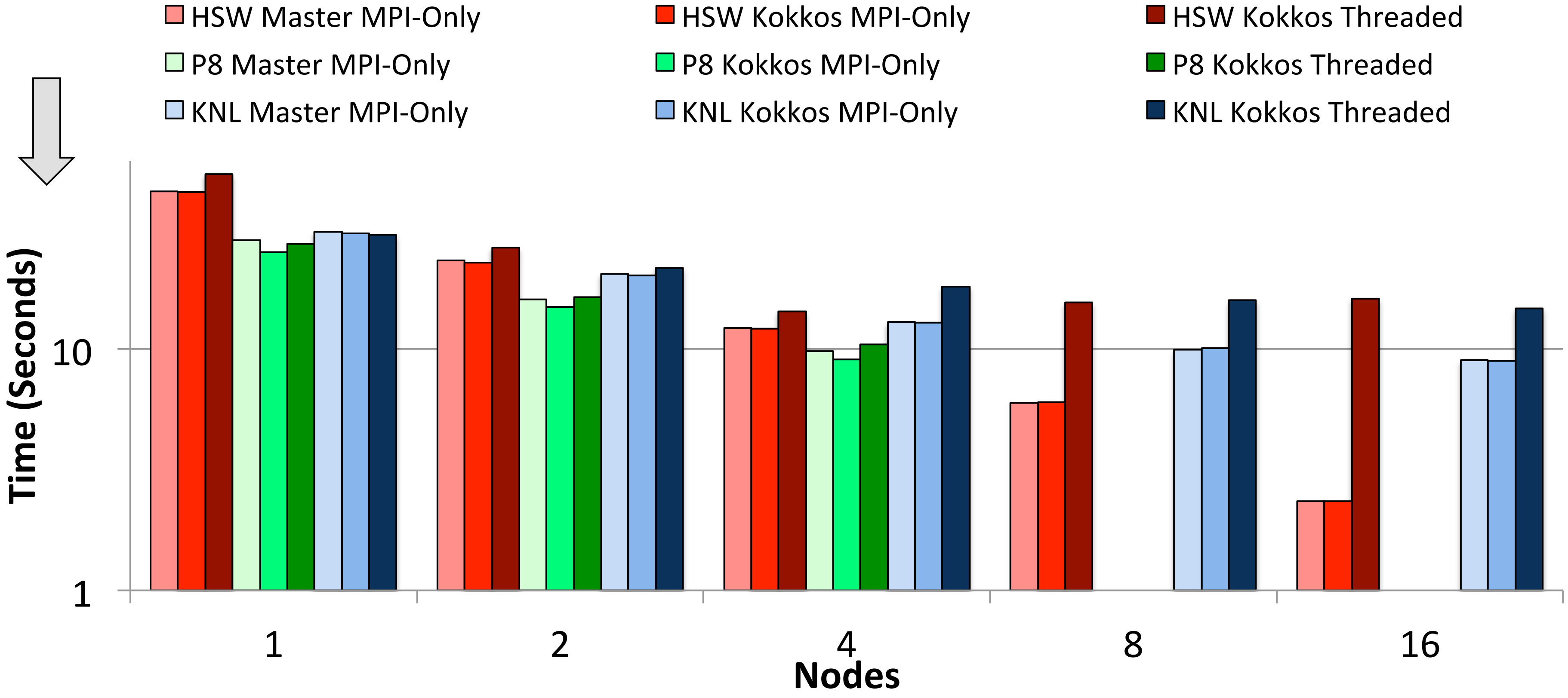


- Optimized the allocation of variables within the assembly
  - Essentially similar to hoisting these outside of compute intensive loops
  - Found because Kokkos prevents users from doing this
- Performance improvement is around 25-40%
- Fix worked back into code master for standard MPI only code and already delivering improvements

NALU Assembly



NALU Solve

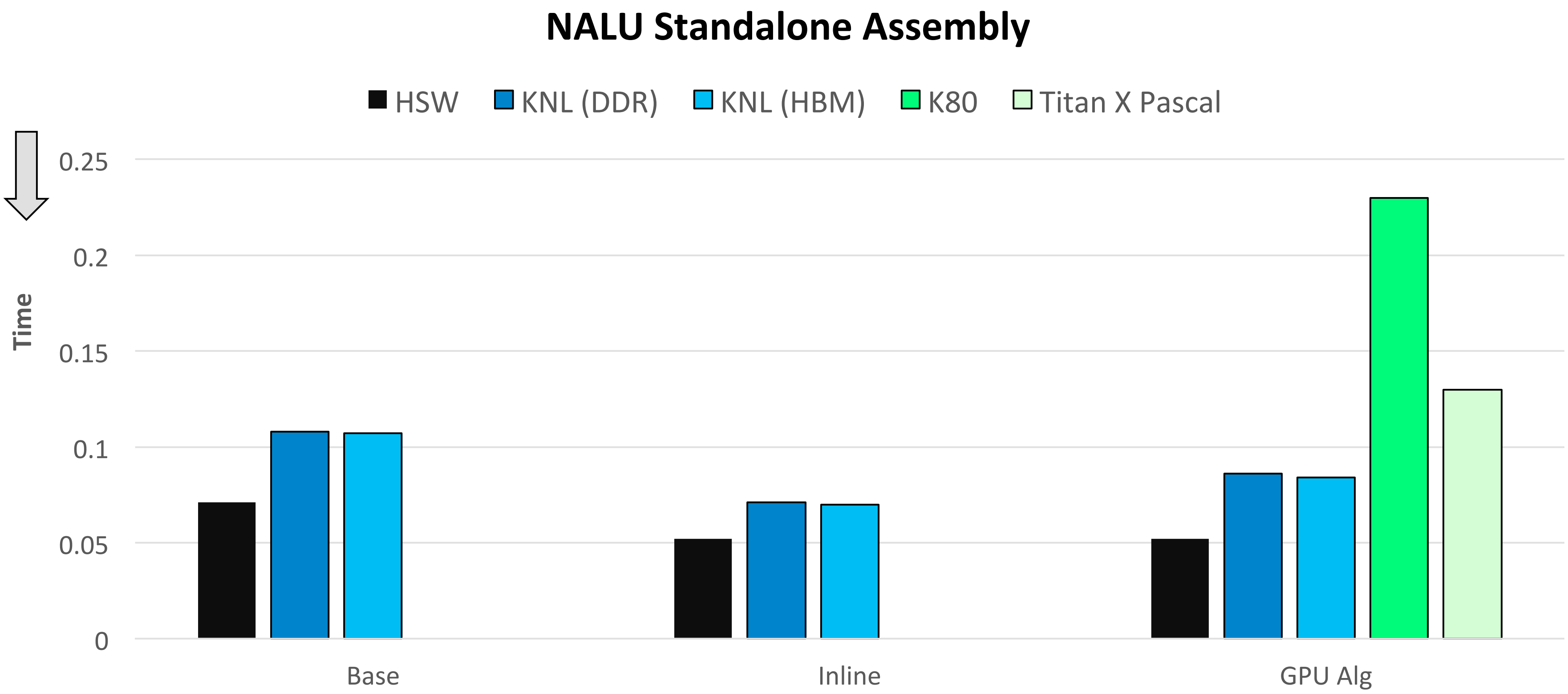

HSW Master MPI-Only
HSW Kokkos MPI-Only
HSW Kokkos Threaded
P8 Master MPI-Only
P8 Kokkos MPI-Only
P8 Kokkos Threaded
KNL Master MPI-Only
KNL Kokkos MPI-Only
KNL Kokkos Threaded
Time (Seconds)
10
1
1
2
4
8
16
Nodes

Exploring Vectorization and GPUs

- **Explore algorithms/implementation/data structure in stand-alone assembly**
  - Binary dumped all input and output arrays to the kernel
  - Read them in, recompute output and compare to gold values

- Base: Basically like the NALU Kokkos code
- Inline: remove Fortan, and inline physics
  - Couple dead-ends explored for example putting in an explicit vectorization level over elements in bucket (i.e. split the element loop into two, one with hardcoded 16)
- GPUAlg: Interleave Scratch Arrays to get coalesced access on GPUs

Standalone NALU Assembly Test Case



Why are GPUs bad at this?

- **Fundamental Problem:** Too little cache on GPUs
  - Require 7 kB temporary space per element in flight
  - On Haswell/KNL/Power: one element per thread
  - On GPUs: one element per ”vector lane”

?

How do we expose more parallelism?

	HSW	KNL	K80 (Kepler)	GP102 (Pascal)
Aggregate (Data) L1 in kB	2*16*32 = 2048	68*32 = 2176	13 * (16+112) = 1664	56 * (24+64) = 4928
Per Thread L1 in kB	16	8	8	11
Per Vector Lane L1 in kB	4	1	0.25	0.34
HBM Bandwidth (Measured by <y|Ax>)	120 GB/s	360 GB/s	165 GB/s	360 GB/s

BROADENING THE IMPACT

Broadening the Impact

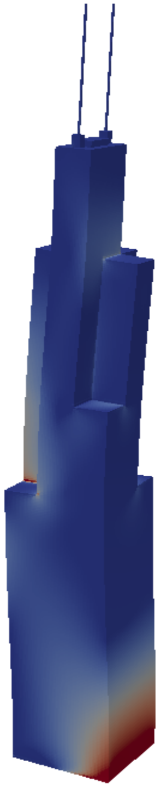
- **Reality is that we see these patterns *a lot* in our code portfolio**
  - Means we can try to reuse these patterns in multiple codes
  - One of the advantages of a “framework” approach to our code base

- So we have been working with various code groups to look at issues which are either similar or relate to this kind of kernel design
  - SIERRA and ATDM primary focus

SIERRA/SM (“Adagio”)

- **Enable the following SIERRA/SM capabilities on GPU (with Kokkos):**
  - Linear beam element
  - Fixed displacement BC
  - Gravity BC
  - Results Output
  - Supporting core algorithms (i.e. central difference time integration)

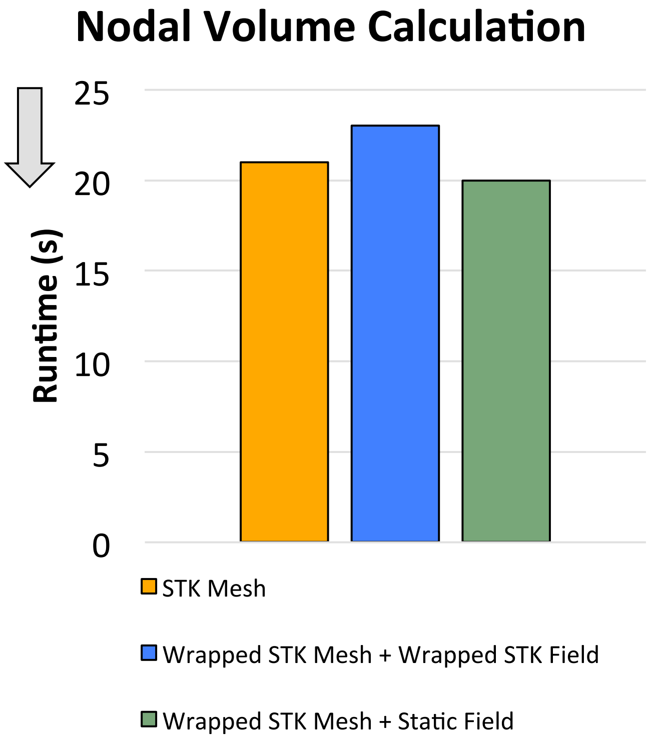
- **Results:**
  - Serial CPU runtime (X86 Sandy Bridge): 366 minutes
  - CPU + GPU runtime (Haswell + K80): 17 minutes
  - Speedup is 22X for **compute** kernels (data movement still hurts)



Work by: SIERRA/SM Team Nate Crane, Mark Mereweather, Mike Tupek, Kendal Pierson

SIERRA STK Team (Mesh Structures)

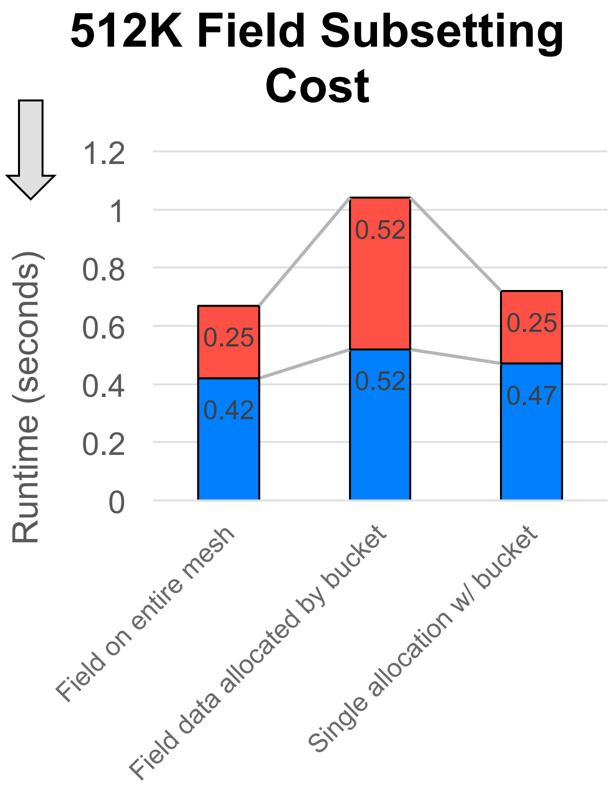
- STK mesh structures are pervasive in SIERRA applications
- SIERRA STK team exploring “mesh wrappers” which insulate code teams from future changes
  - Need to be “low overhead”
  - Easy to integrate
  - Opportunity to simplify some internals
- Prototype single node evaluations underway
- Exploring with Kokkos on CPU, GPU, KNL *etc.*



Work by: SIERRA/STK Team (now led by Kendal Pierson)

SIERRA STK Team (Mesh on GPU)

- **Focused sprint on using STK meshes on GPU w/ Kokkos**
- Different types of data structure layouts
  - Utilize different types of Kokkos-Views underneath
  - Different iteration schemes and mesh auras
- Different performance characteristics as a result
- Finding more consistent timings across GPU
- Still part of the STK "Learning" product for additional studies



Work by: SIERRA/STK Team (now led by Kendal Pierson)

SIERRA/TF Aria

Sandia National Laboratories

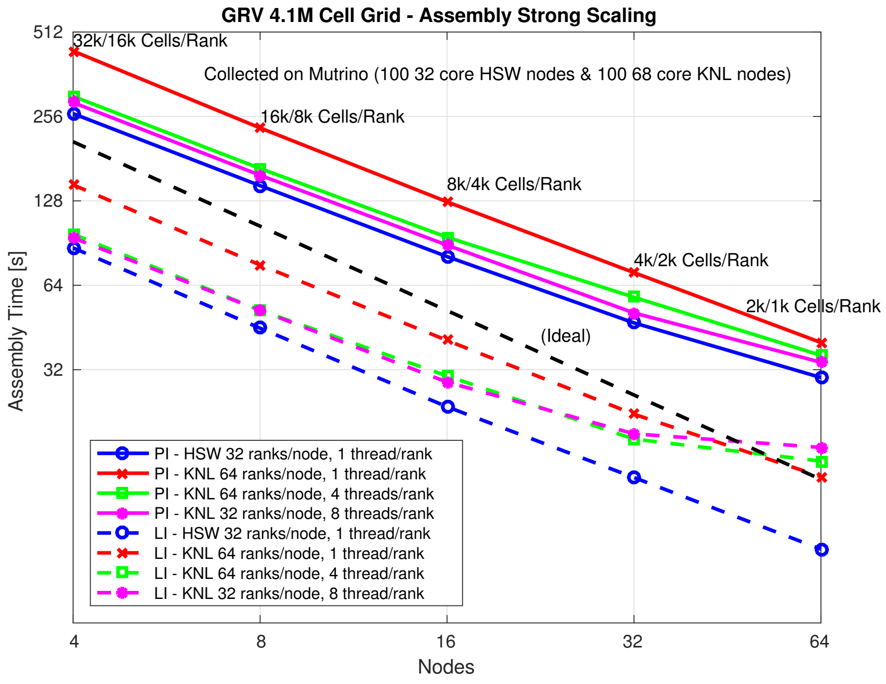
- Identified area of weakness in SIERRA/TF team room discussions
- **Expression system in Aria is a complicated area which is not well represented in any existing mini-app**
  - Extracted core Expression system used for matrix assembly in Aria into non-export controlled Ariamini.
  - Serves as a smaller test bed for determining how to integrate Kokkos for thread-parallel matrix assembly in Aria that can be shared with Trilinos+Kokkos developers in 1400
  - Only dependency is Trilinos (STK)
- Plan to use Ariamini as a new codesign vehicle during the next FY

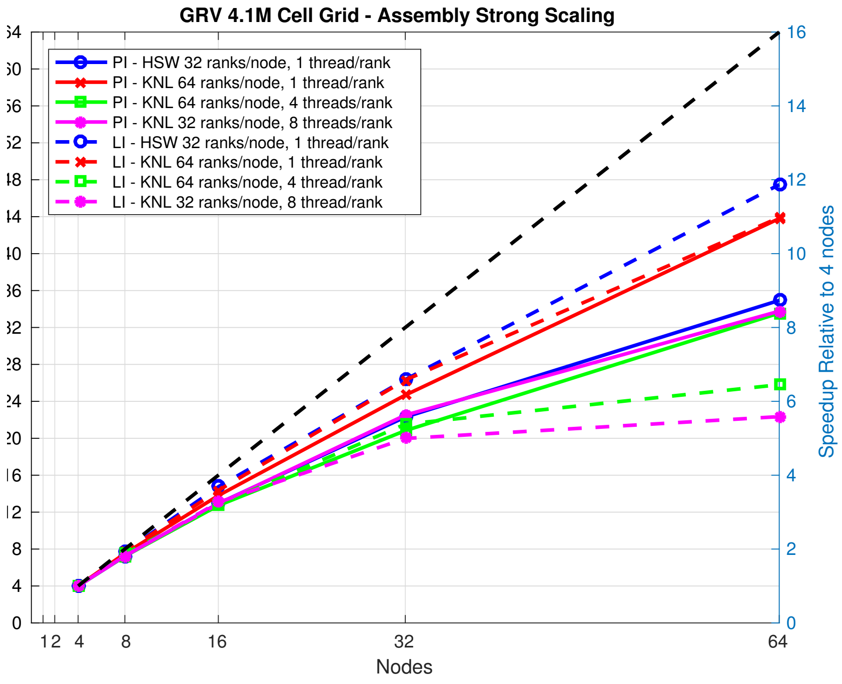
Work by: SIERRA/TF Team, Jon Clausen and Victor Brunini

ATDM/SPARC (Assembly)

Work by: Micah Howard

Sandia National Laboratories

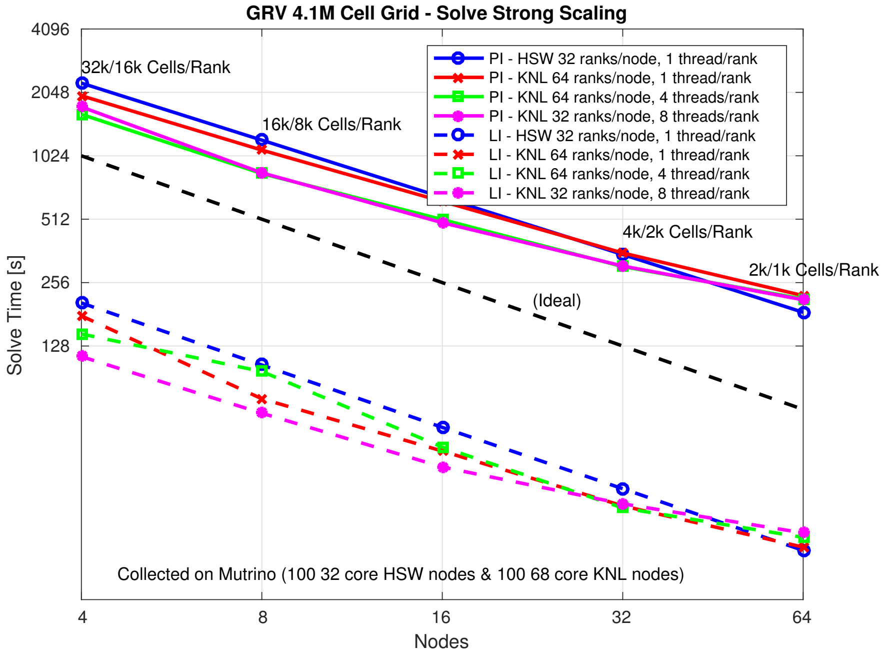


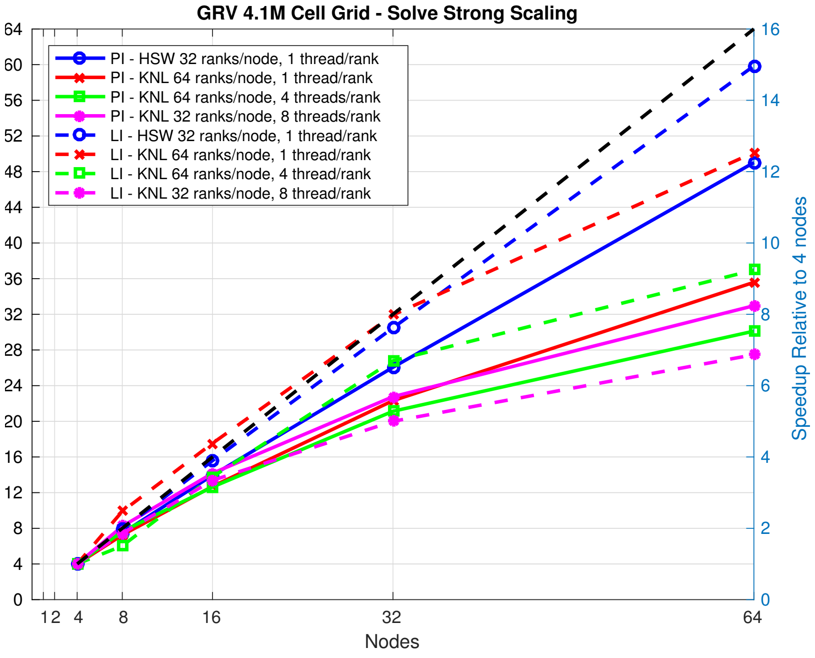


Still the case that HSW out performs KNL on Assembly (mini-apps and proto experiences still correlate)

ATDM/SPARC (Solve)

Work by: Micah Howard and Andrew Bradley



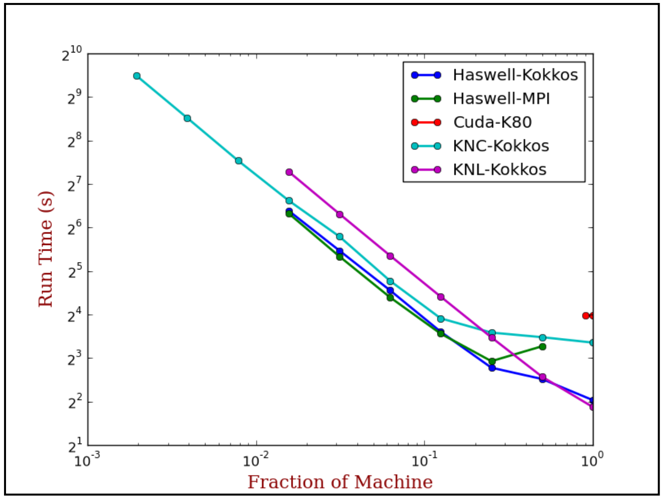


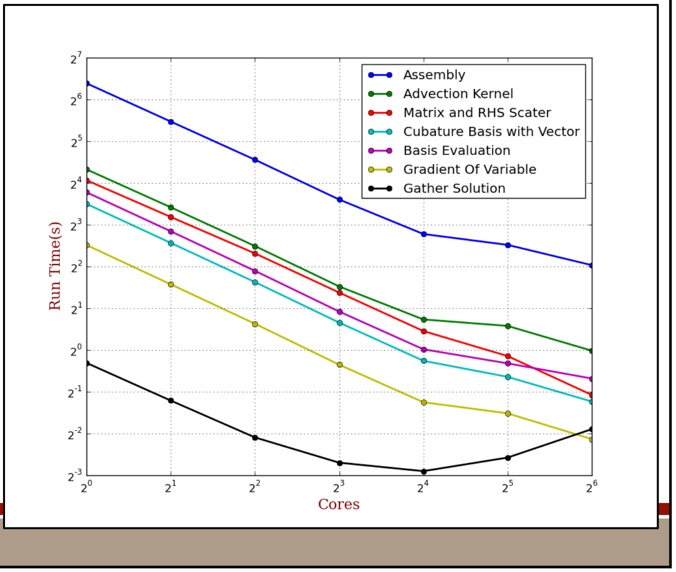
KNL now faster for solve phase although scaling needs to be improved, strong thread performance

ATDM/EMPIRE

Work by: Matt Bettencourt

- PIC-based code developed as a future capability for RAMSES suite
- Written using Kokkos and Trilinos entirely
- Works entirely on CPU, KNC, KNL and GPU
- MiniPIC variant (including Kokkos and Trilinos) used for APEX/ATS-3 procurement
  - Results show similar performance for assembly and solve
- Significant focus this FY on analyzing kernel performance using the Kokkos profiling hooks





Takeaways and Discussion

- Continued development and refinement of abstractions in code to permit greater portability (see these are usually relatively cheap)
- Significant diversity in how applications teams/developers want to evaluate performance (rank-for-rank, node-for-node, *etc.*)
  - Need to set clear expectations for what is to be expected
- Still very diverse set of benchmarking platforms used so not always easy to get a definitive cohesive picture
  - Application Performance Team at SNL working hard to address this issue
  - Greater availability of test resources will help
- **Stronger engagement in FY16 with much broader interest and activities at SNL**

SUMMARY AND THOUGHTS

Meeting the Milestone Requirements…

✓

> *"Improvements from proxy applications from each lab have been identified and evaluated for applicability in IC or ATDM codes"*

- Showed connections from our production applications, mini-apps and proto-apps to some issues we see in other production applications and broad range of platforms
- Used mini-apps as a testing and bring up environment for NGP systems
    - And yes, they have solved issues which ***do*** impact our codes
- Showed that we do observe some weaknesses but also that our mini-apps are a huge aid in our prototyping, our proto-apps are filling some of the gaps
- Seeing our application teams realizing value of mini-apps and creating their own – many are more complex that our initial output into Mantevo

Meeting the Milestone Requirements…

✓

"The team has reported how the proxy applications are representative and where they could be improved. "

- Already working on a number of fixes and newer mini-apps that can be used as exemplars for more complex issues
- Pushing some changes to Github Mantevo (ready for SC)
- Seeing new mini-apps from application teams as a way to experiment with programming models and data structure changes (Ariamini and STK)
- Also seeing where these can direct our platform bring ups/environments

So how are we doing?

- **Clear we have been some significant steps forward on getting initial code ports onto new platforms and tool chains**
  - Historically this has been a huge problem and long delays
  - Seeing wide variety of issues with environment, compilers, code, libraries
  - Mini-apps have really helped us to work on basic problems and base performance metrics

- Substantial use of Kokkos and growing use of Trilinos functions across application portfolio
  - Seeing this drive new features, greater performance focus
  - Many issues relating to numerical reproducibility and variances between platforms

- Much stronger engagement with application community

What are we struggling with?

- Still long lead times to see fixes from vendors coming into production
  - This is getting better but it could also be improved (CUDA in particular)

- Effects on numerics of FMA, vectorization are not always consistent
  - Create concern across our code teams that we may have bugs
  - Use of atomics and threads exacerbate this and create further worry

- Compile and link times continue to remain a problem for developer productivity
  - Very significant problem on POWER8 with XL compiler but see elsewhere

Working with Vendors

- **FY16 has seen huge engagement with vendors on compilers and software stack**
  - Intel, IBM, NVIDIA and PGI
  - Large number of bugs and performance issues reported (some fixed)

- Shared significant performance studies and information with Intel, IBM and some with NVIDIA
  - Presented many results at conferences and in private feedback to vendors

- Seeing strong collaboration with ARM on numerical libraries area using our ARM test beds

Coming Year…

Sandia National Laboratories

- **Focus on improving our KNL and GPU performance**
  - KNL environment beginning to strengthen
  - POWER8+/GPU should be easier with arrival of Pascal/NVLINK systems
- Emerging look at ARM64 systems
  - Already underway but still basic environment at this stage
- Application shift to support more detailed analysis within ATDM project as codes come online
- Strong IC engagement as porting ramps up

Thank You…

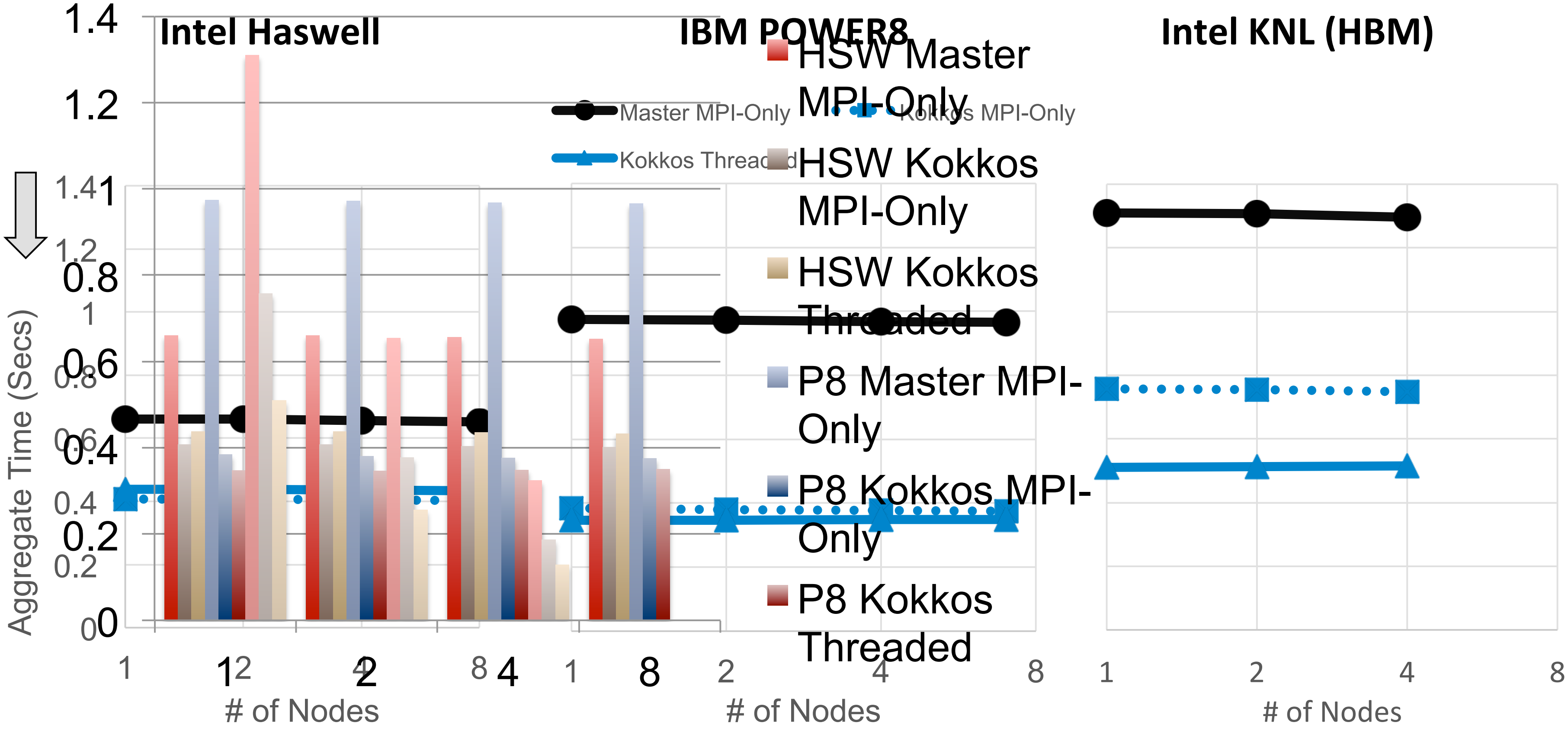
- Sandia Application Performance Team (“APT”)
- ASC Advanced Test Bed Team
- SIERRA, RAMSES and CTH Code Groups
- Trilinos Developers
- ATDM Code Groups
- ATDM Software Environment Team

- L2 Milestone Review Team!

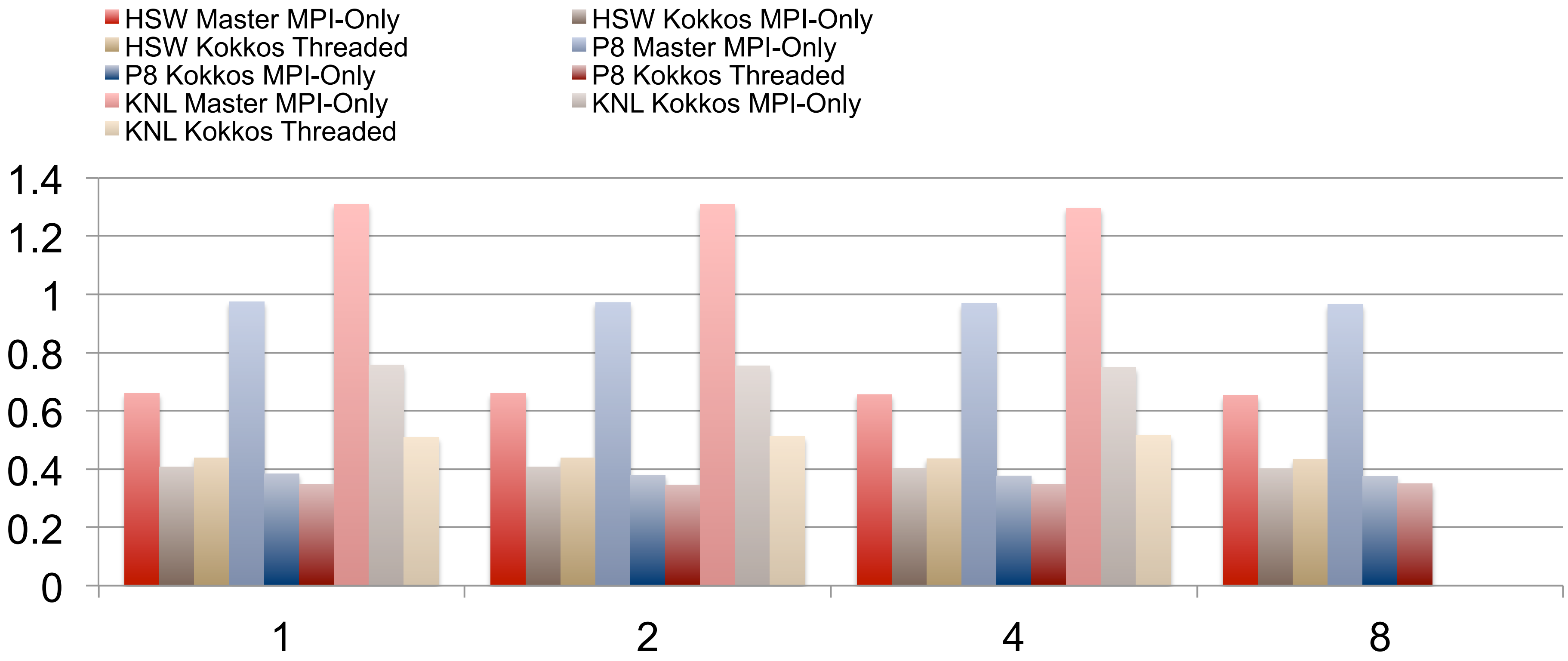
Sandia National Laboratories

- Wall time versus CPU time
- Link times horrendous
- Compiler bugs
- FMA still a problem?

Assembly in NALU



Sandia National Laboratories


HSW Master MPI-Only
HSW Kokkos MPI-Only
HSW Kokkos Threaded
P8 Master MPI-Only
P8 Kokkos MPI-Only
P8 Kokkos Threaded
KNL Master MPI-Only
KNL Kokkos MPI-Only
KNL Kokkos Threaded