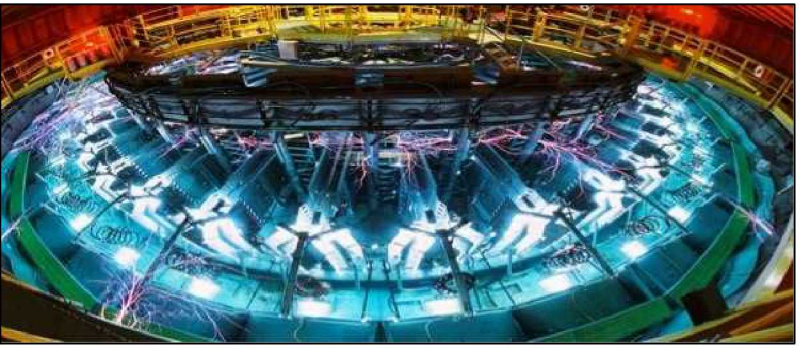
SAND2015-7327PE



Codesign at Sandia: LULESH and MiniAero

J. Cook, H.C. Edwards, D. Dinge, M. Glass, S.D. Hammond, R. Hoekstra, P.T. Lin, M. Rajan, C.R. Trott and C.T. Vaughan
[sdhammo | crtrott] @sandia.gov

Application Performance Team
Center for Computing Research
Sandia National Laboratories, NM, USA

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.

EXECUTIVE SUMMARY

Introduction

- In order to have high performance of production codes on future ATS systems we will need to add on-node parallelism
- Current options include using OpenMP directives or writing directly to native programming models like CUDA, pthreads *etc*
- But .. we also want a single code base to work across machines to decrease development costs and make debugging, maintenance easier *etc*
- For C++ codes there is the possibility of using language features to provide an abstraction of parallel kernels which *can* be made to run on multiple hardware types efficiently
- Code teams want to understand want the tradeoff is between a directives based solution and the proposed C++ language abstractions (tradeoff in performance, portability and programmer productivity)

Work Performed in this Milestone (Part 1)

- Spirit of the milestone is to take one mini-app from another lab and demonstrate that this can be used with our local (SNL) programming model Kokkos

- We took the LULESH hydrodynamics mini-app from LLNL and rewrote/optimized it using a variety of programming models including directives, Kokkos and the LLNL developed RAJA

- Sandia was able to utilize the ASC Adv. Arch. Testbeds to show performance on multi-core, many-core and GPU architectures

- Evaluation of programmer productivity in terms of changes required to basic serial source code to add parallelism

Work Performed in this Milestone (Part 2)

- We were also required to analyze one of our own locally developed mini-apps in Kokkos for performance and portability

- Sandia was again able to show efficient execution using the MiniAero Kokkos implementation on multi-core, many-core and GPU architectures

- There are two basic algorithms in MiniAero (atomics and alternatively, gather/sum). Our analysis shows no clear winner, best performance dependent on architecture
  - Implies we may not always be able to have a single algorithm in a single source to run everywhere

- Used MiniAero to perform basic analysis for what we might get on the Trinity Phase-I Haswell partition and some limited analysis of Phase-II KNL vectorization

ASC L2 Tri-Lab Codesign Milestone for 2015

Level: 2	**Fiscal Year**: FY15	**DOE Area/Campaign**: ASC
Completion Date: 9/30/15		
ASC nWBS Subprogram: ATDM, CSSE, IC		
Participating Sites: LLNL, LANL, SNL		
Participating Programs/Campaigns: ASC		

Description: This milestone is a tri-lab deliverable supporting the ongoing co-design efforts in the program (IC & CSSE) as well as the new ATDM activities. In FY14, a milestone evaluating the performance and underlying bottlenecks of key proxy applications on advanced architecture test-beds or AT systems was completed. In addition, each lab has been developing and exploring promising new parallel programming models that will provide abstractions for performance portability, especially at the node level. This milestone focuses on building upon those [illegible] through a combination of internal performance improvements, deepe[illegible]tion of abstractions, and external codesign influence.

Each lab will choose one or more proxy applications and refactor them through the use of new programming models & tools, algorithms, or DSLs, resulting in either demonstrable speedup, improved portability through abstractions, or as a stretch goal - both. If only one proxy application is chosen, two or more refactored versions will be provided. Improvements will be demonstrated across at least 2 advanced architectures (testbed or ATS). Performance improvements will be relative to proxy apps and related metrics gathered in the FY14 milestone. Programming abstraction demonstrations will use at least one proxy app developed at another laboratory (in collaboration and agreement with that lab) in order to demonstrate broad applicability across variable programming styles.

Successful and unsuccessful attempts will be reported as lessons learned. The tri-la[illegible] co-design project will work closely with the *Forward vendors to make available studied proxy apps and related data (e.g. traces or simulator results) for vend[illegible]

Completion Criteria: This milestone will be completed when:

1. At least two proxy apps (or at least two implementations of one proxy app) from each lab have demonstrated performance improvements or improved portability across two advanced architectures.
2. A report has been completed by the 3 labs detailing lessons learned - both successes and failures, in regards to performance and/or portability improvements.
3. The milestone team has communicated appropriate information including source code, metrics, trace data, and simulator analysis for use in vendor-focused research.

Customer: ASC Application Code Teams

Milestone Certification Method:

A program review is conducted and its results are documented.

Professional documentation, such as a report or a set of viewgraphs with a written summary, is prepared as a record of milestone completion.

Each lab will choose one or more proxy applications and refactor them through the use of new programming models & tools, algorithms, or DSLs, resulting in either demonstrable speedup, improved portability through abstractions, or as a stretch goal – both.

Completion Criteria: This milestone will be completed when:

- *At least two proxy apps (or at least two implementations of one proxy app) from each lab have demonstrated performance improvements or improved portability across two advanced architectures.*
- *A report has been completed by the 3 labs detailing lessons learned - both successes and failures, in regards to performance and/or portability improvements.*
- *The milestone team has communicated appropriate information including source code, metrics, trace data, and simulator analysis for use in vendor-focused research.*

Path Forward

- In FY16 we will be looking to use the lessons learned from FY15 to begin supporting SIERRA code groups in preparation for Trinity Phase-I and Phase-II

- We will be providing best practices for the introduction of on-node parallelism including OpenMP and Kokkos

- We will be using the analysis and profiling technologies used from the FY15 milestone to help provide insight into application bottlenecks and areas of poor scalability

Sandia National Laboratories

ASC L2 CODESIGN MILESTONE REVIEW PRESENTATION

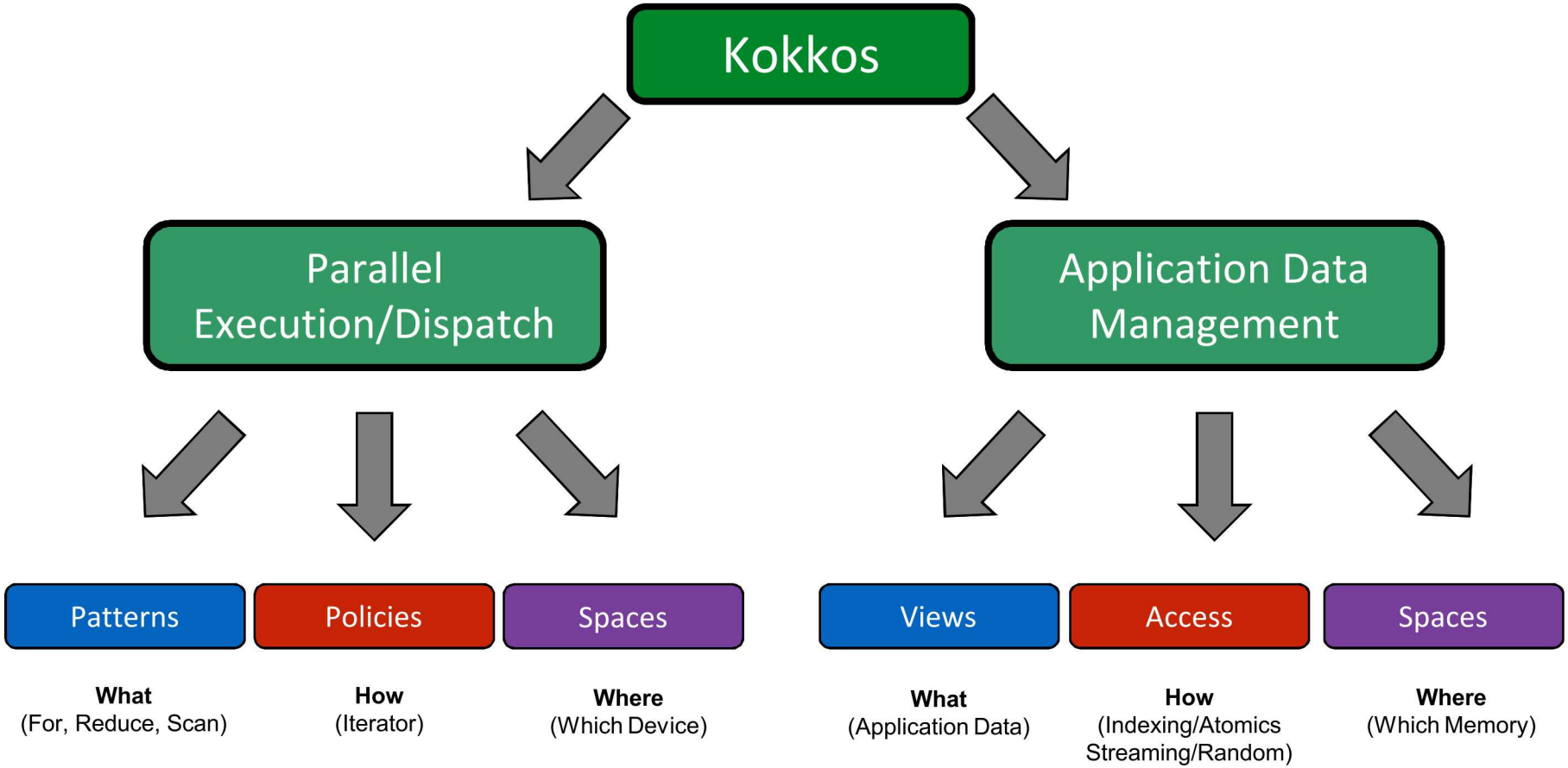
Overview of SNL

- **Part I: Performance, Portability and Productivity of C++ Abstractions for the LULESH mini-app**
  - Overview of our porting activities
  - Comparison of performance on leading HPC architectures for OpenMP, RAJA and LULESH
  - Evaluation of programmer effort required for OpenMP, RAJA and Kokkos
- **Part II: Performance Analysis of MiniAero**
  - Comparison of Scaling (MPI/OpenMP) for Haswell, BlueGene/Q, Knights Corner and NVIDIA K80 GPUs
  - Initial expectations for codes on Trinity Phase-I and Phase-II
- Discussion

Sandia National Laboratories

PORTING LULESH TO KOKKOS

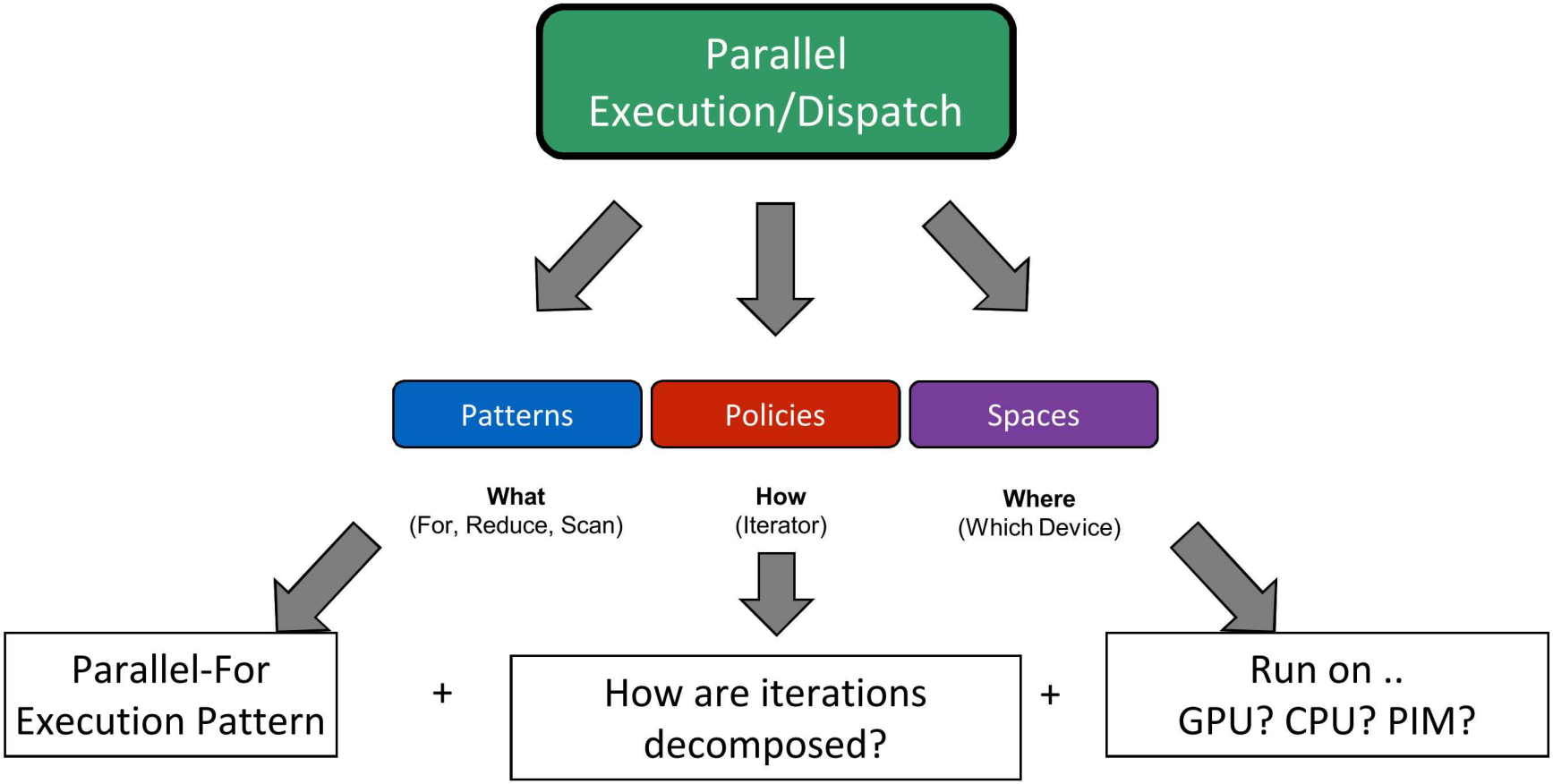
Kokkos Programming Model



Separation and Abstraction of Concerns

Abstract Application Data and Computation

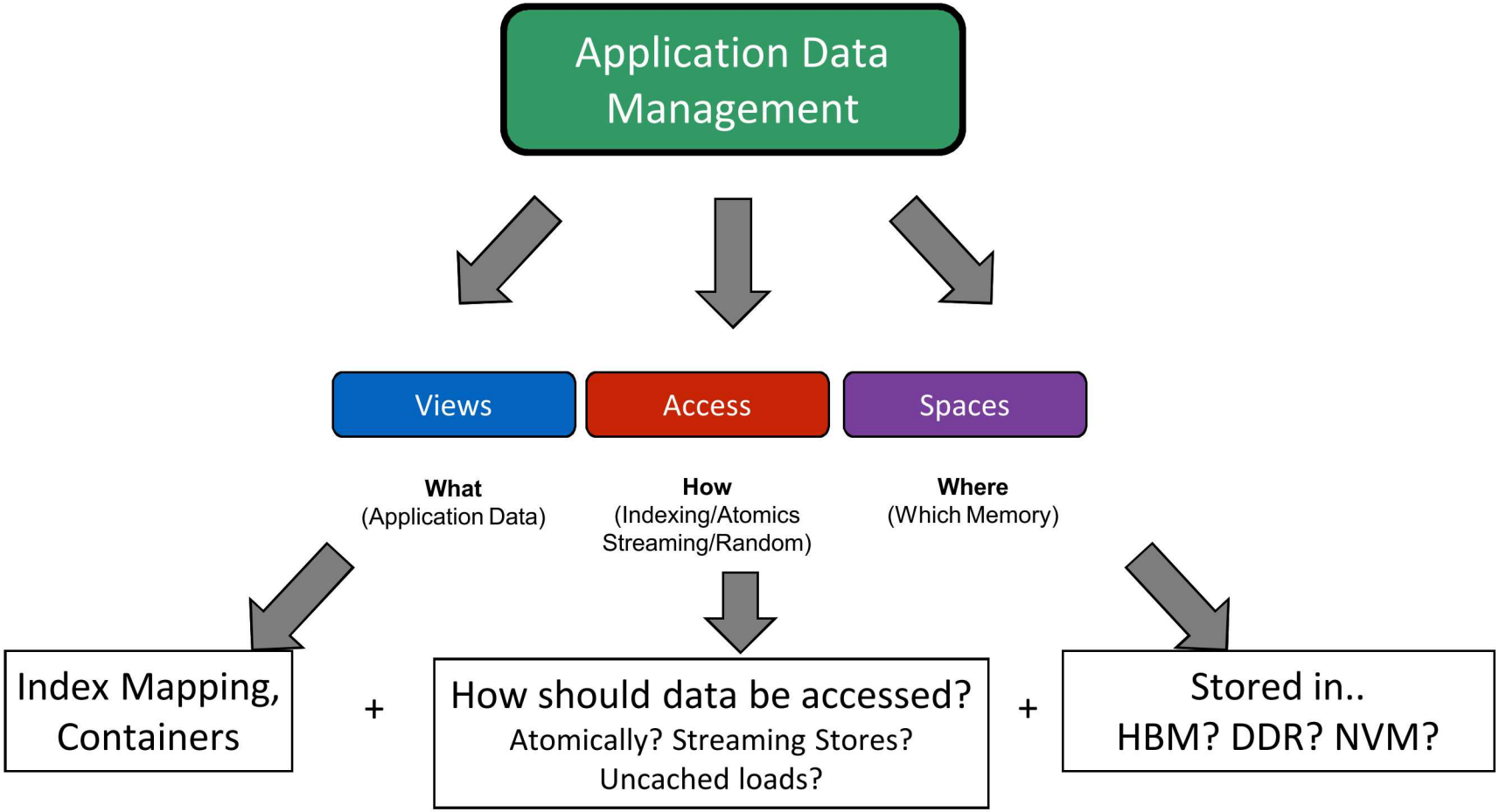
Kokkos Programming Model (Compute)



Sensible defaults for many execution spaces to reduce programmer overhead

Let Sandia research and Kokkos developers handle the heavy work

Kokkos Programming Model (Data)



Sensible defaults for many memory spaces to reduce programmer overhead

Let Sandia research and Kokkos developers handle the heavy work

What Does Kokkos Run on Today?

Kokkos is running on every advanced architecture test bed, prototype option on AMD systems

ASC Trinity Phase I – ATS1

- ✓ Intel Xeon Haswell (Intel, GNU, LLVM)

ASC Trinity Phase II – ATS1

- ✓ Intel Xeon Phi Knights Landing Emulator (Intel)

ASC Sierra – ATS2

- ✓ POWER8 (XL, GNU)
- ✓ NVIDIA GPU (K20, K40, K80, NSDK-7.5)

ASC TLCC-2

- ✓ Intel Xeon Sandy Bridge (Intel, GNU, LLVM, Cray)

ASC Advanced Arch. Test Beds

- ✓ AMD Kaveri APU (GNU-HSA)
- ✓ ARM64 (GNU, LLVM)
- ✓ Intel Xeon Phi Knights Corner (Intel)

✓ = Kokkos Build Type in Release ✓ = Prototype/Research

Examining Porting Strategies for Code Teams

- **Very large proportion of ASC code at Sandia is MPI only**
  - Implies a serial on-node model with limited thread safety applied

- Starting point for this study is the **serial** version of LULESH
  - Taken from the OpenMP version but with all OpenMP pragmas, reductions and specializations removed ("proxy" for "real" code)

- Provide several implementations to evaluate metrics:
  - **Kokkos:** Minimal CPU, Minimal CPU with ref lambdas, Minimal GPU, Optimized-V1, Optimized-V2, Optimized-V3
  - **OpenMP:** Original OpenMP from LLNL, Optimized OpenMP from SNL
  - **RAJA:** RAJA-Basic and RAJA-Index-Set

Non Kokkos-Variants

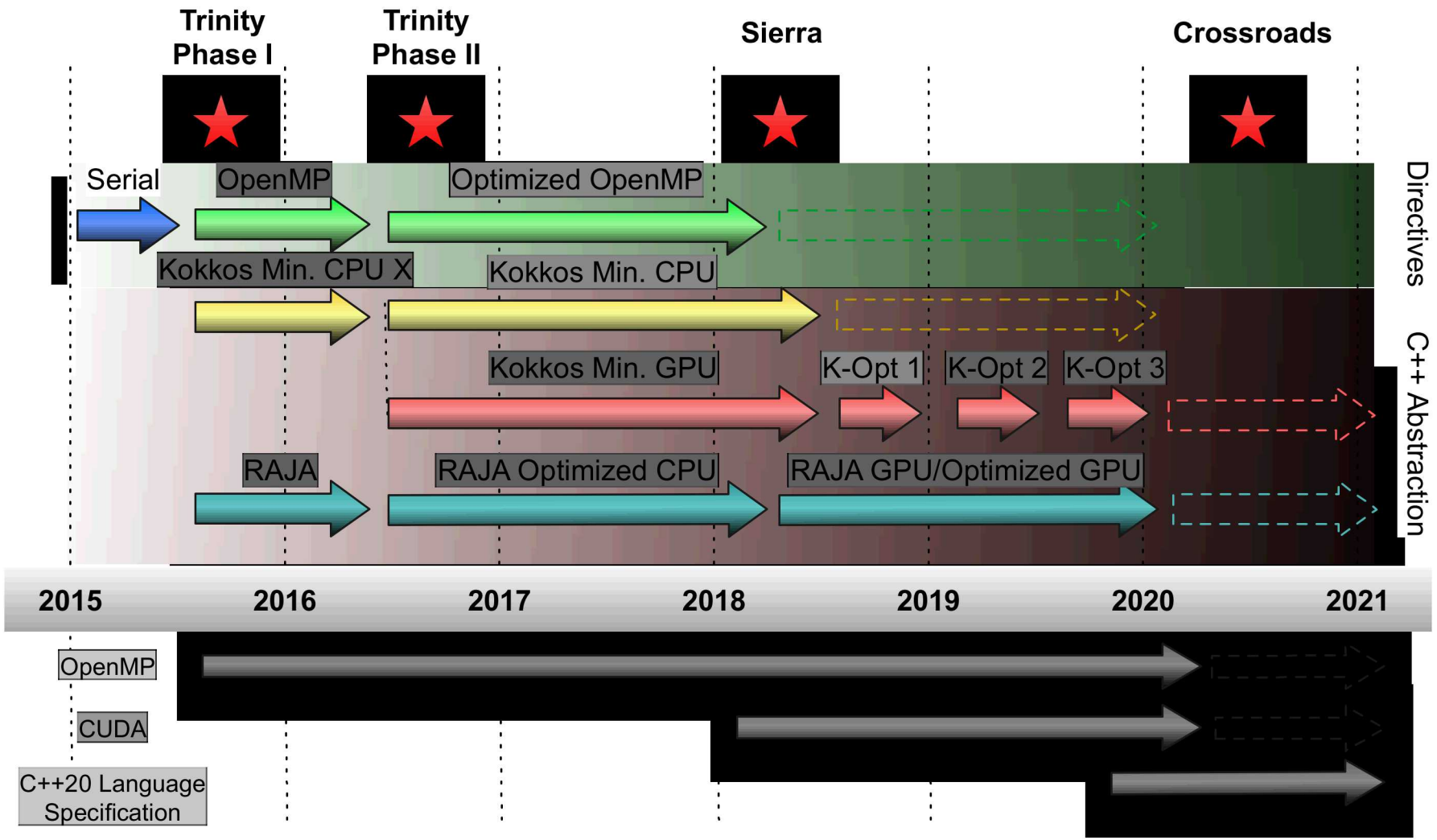
- **RAJA-Basic:** code provided by Jeff Keasler and Rich Hornung from LLNL, uses RAJA abstractions for parallel dispatch
- **RAJA-IndexSet:** code provided by Jeff Keasler and Rich Hornung from LLNL, uses RAJA abstractions for data iteration
- **OpenMP Original:** NO-RAJA variant from LLNL
- **OpenMP Minimal:** a stripped down version using basic parallel-for schemes and atomic operations developed from serial using Intel AdvisorXE and InspectorXE (akin to developer using tools)
- **OpenMP Optimized:** Sandia optimized version which improves vectorization and reduction performance

Optimized Kokkos Variants

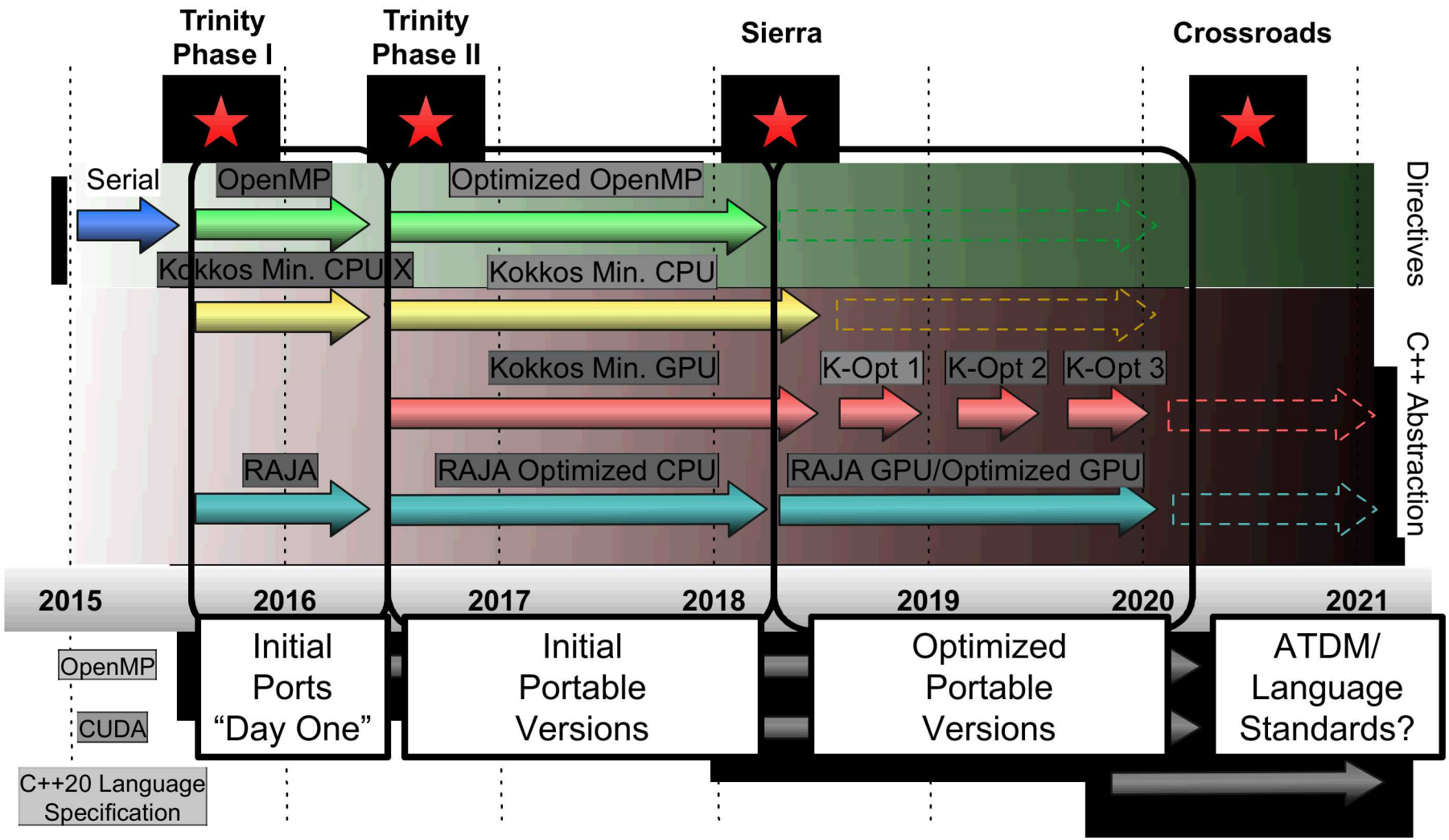
Sandia National Laboratories

- **Kokkos-Minimal-CPU:** developed by a physicist with limited experience writing threaded code (our experiment for code we would get from many code groups)
- **Kokkos-Minimal-CPU-RL:** basic port to Kokkos which utilizes capture-by-reference lambdas to significantly decrease programmer burden
- **Kokkos-Minimal-GPU:** extension of Kokkos-Minimal-CPU to work on the GPU (mainly data structure const changes)
- **Kokkos-Optimized-v1:** eliminate buffer realloc; reduce register pressure
- **Kokkos-Optimized-v2:** use Kokkos Views with Layout and Traits, Hierarchical Parallelism
- **Kokkos-Optimized-v3:** kernel fusion

Swim Lanes for Code Teams

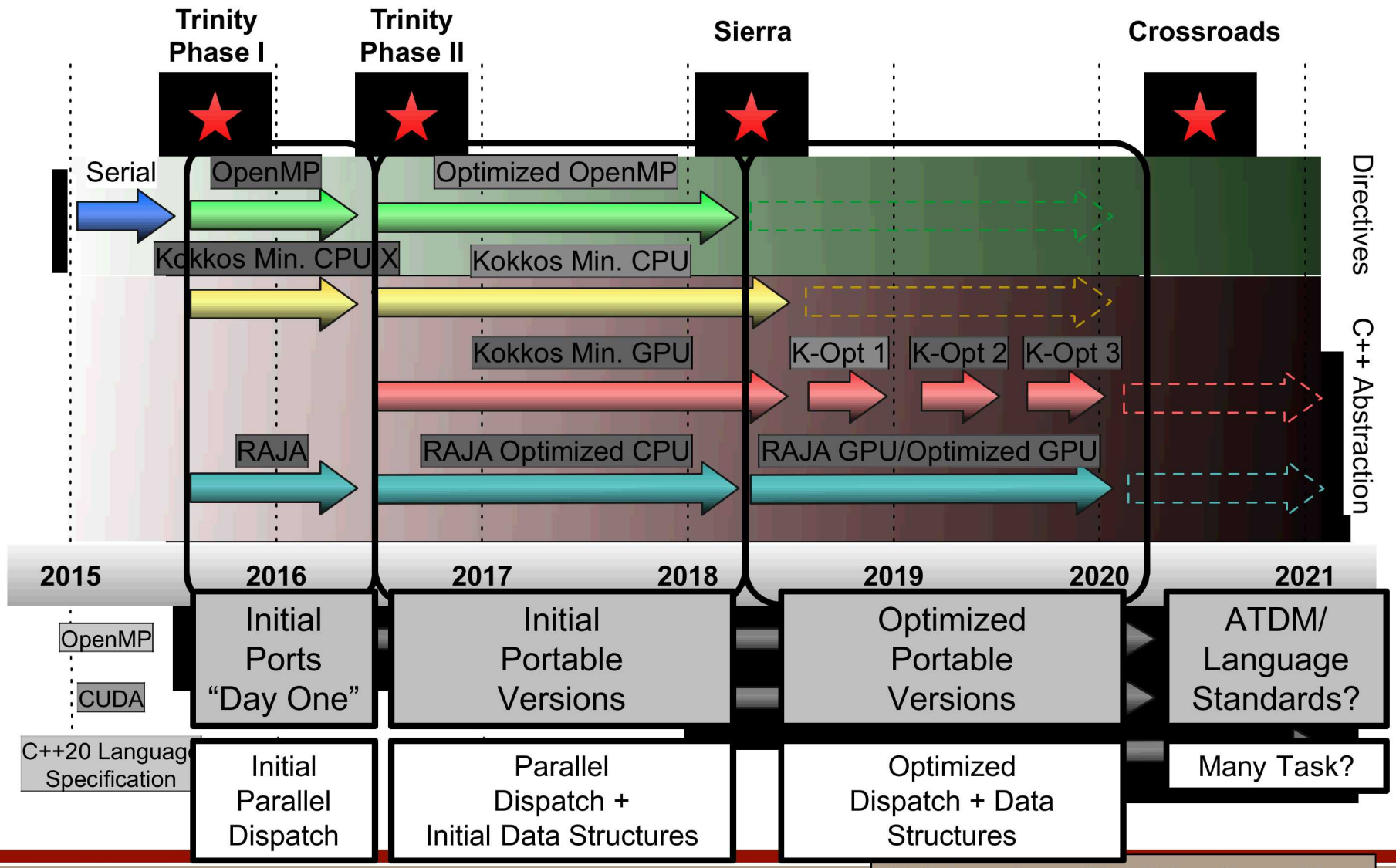


Swim Lanes for Code Teams



This is not an official Sandia position

Swim Lanes for Code Teams



This is not an official Sandia position

What are We Presenting?

- In an ideal world we would have all code ported with minimal changes
  - Very unlikely to happen for ASC codes, complicated, legacy algorithms, years of engineering

- **So what can we hope for?**
  - Progression of modifications to the code to get them ready for NGP
  - Initial ports require less modification to get code up and running but don't give top performance
  - Slowly evolve code/data-structures to give better cross-platform performance

- Sandia ASC L2 results show what we might be able to expect in a small case study using LULESH
  - We think there is a similar story for Kokkos and RAJA

Sandia National Laboratories

Evaluating Performance Across Architectures

PERFORMANCE PORTABILITY OF LULESH VERSIONS

ASC Arch. Test Bed Systems Used For Testing

- **Shepard Intel Haswell**
  - Dual-socket, 16-cores/socket, 2 x 256-bit FP-FMA SIMD/core, SMT-2
  - 128GB RAM/socket
  - Intel 15.2.164 Compiler with OpenMPI 1.8.X
- **Compton Intel Sandy Bridge and Knights Corner**
  - Dual-socket 8-cores/socket, 2x256-bit FP SIMD/core, SMT-2
  - 32GB RAM/socket
  - Intel 15.2.164 Compiler with OpenMPI 1.8.X (Sandy Bridge)
  - 57-core KNC-C0, 1.1GHz, 6GB/RAM
  - Intel 15.2.164 Compiler with Intel MPI 4.1.036 (KNC)

ASC Arch. Test Bed Systems Used For Testing

- **White POWER8**
  - Dual-socket, Dual-NUMA/socket POWER8, 3.4GHz
  - 5-cores/NUMA = 10 cores/socket = 20 cores/node, SMT-8/core
  - 128GB RAM/NUMA = 512GB/node
  - GNU 4.9.2 with OpenMPI 1.8.X
  - IBM XL 13.1.2 with OpenMPI 1.8.X

- **Hammer APM ARM-64/v8**
  - Single socket/node, 8-cores/node, 2.4GHz
  - 32GB RAM/socket
  - GNU 4.9.2 with OpenMPI 1.8.X

ASC Arch. Test Bed Systems Used For Testing

- **Shannon Intel Sandy Bridge + NVIDIA Kepler K40/80**
  - Dual-socket, 8-cores/socket Sandy Bridge = 16 cores/node
  - 32GB RAM/socket
  - NVIDIA Kepler K40 per socket
  - NVIDIA CUDA 7.5 SDK
  - GNU 4.7.2 with OpenMPI 1.8.X (compiled with CUDA support)

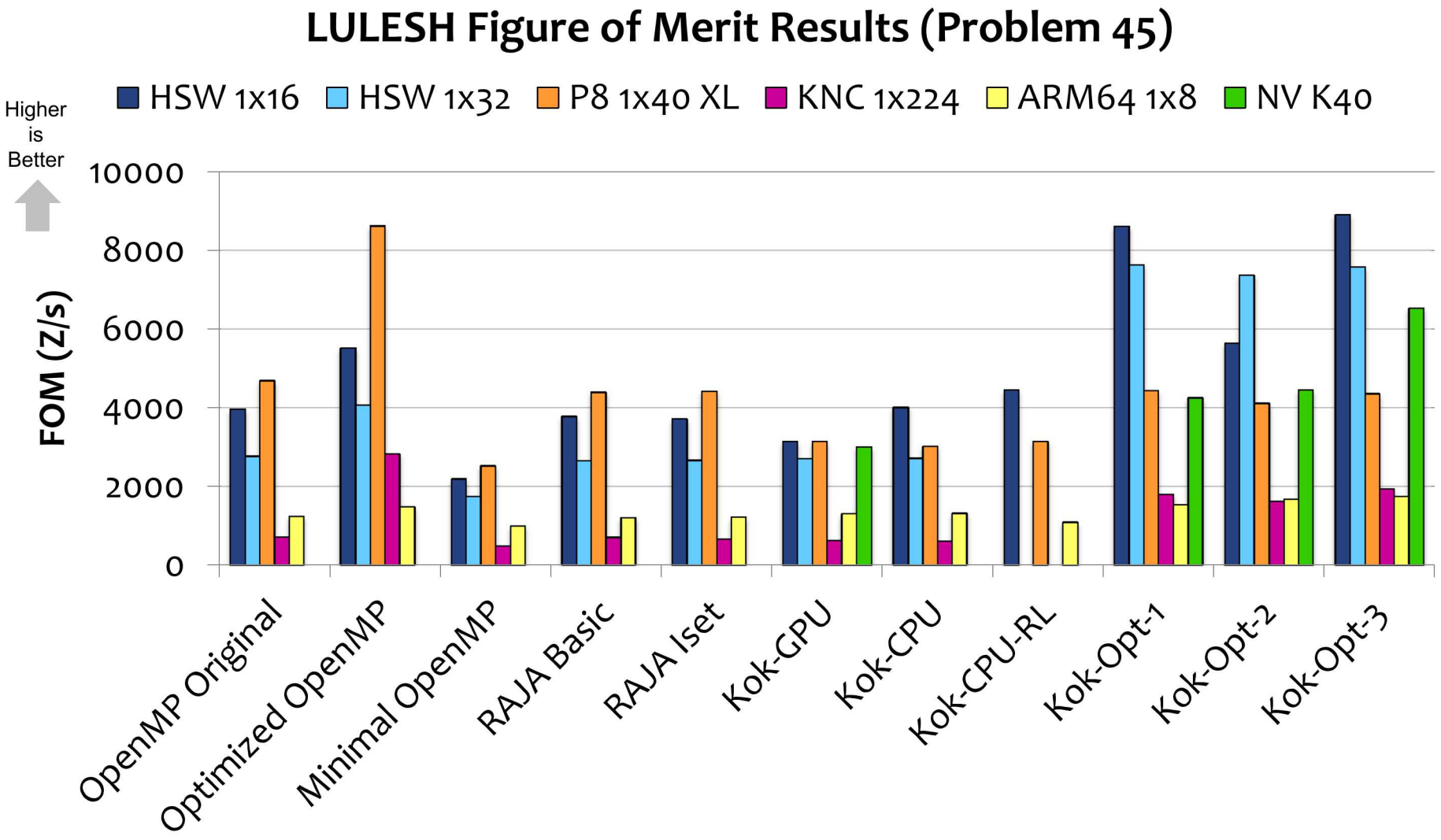
Optimization Notice

- Where possible we have selected architecture appropriate optimization flags to improve performance
  - **Kokkos** – baked into the Kokkos Makefile system
  - **RAJA** – baked into RAJA Makefile system and RAJA header files for alignment, vectorization width *etc* (header additions are annoying)

- Results are the harmonic mean of LLNL-coded "Figure of Merit" (FOM) from a minimum 10 runs, max, min etc are all recorded
  - Error bars are typically very small (1-3%) so are not included in plots for brevity

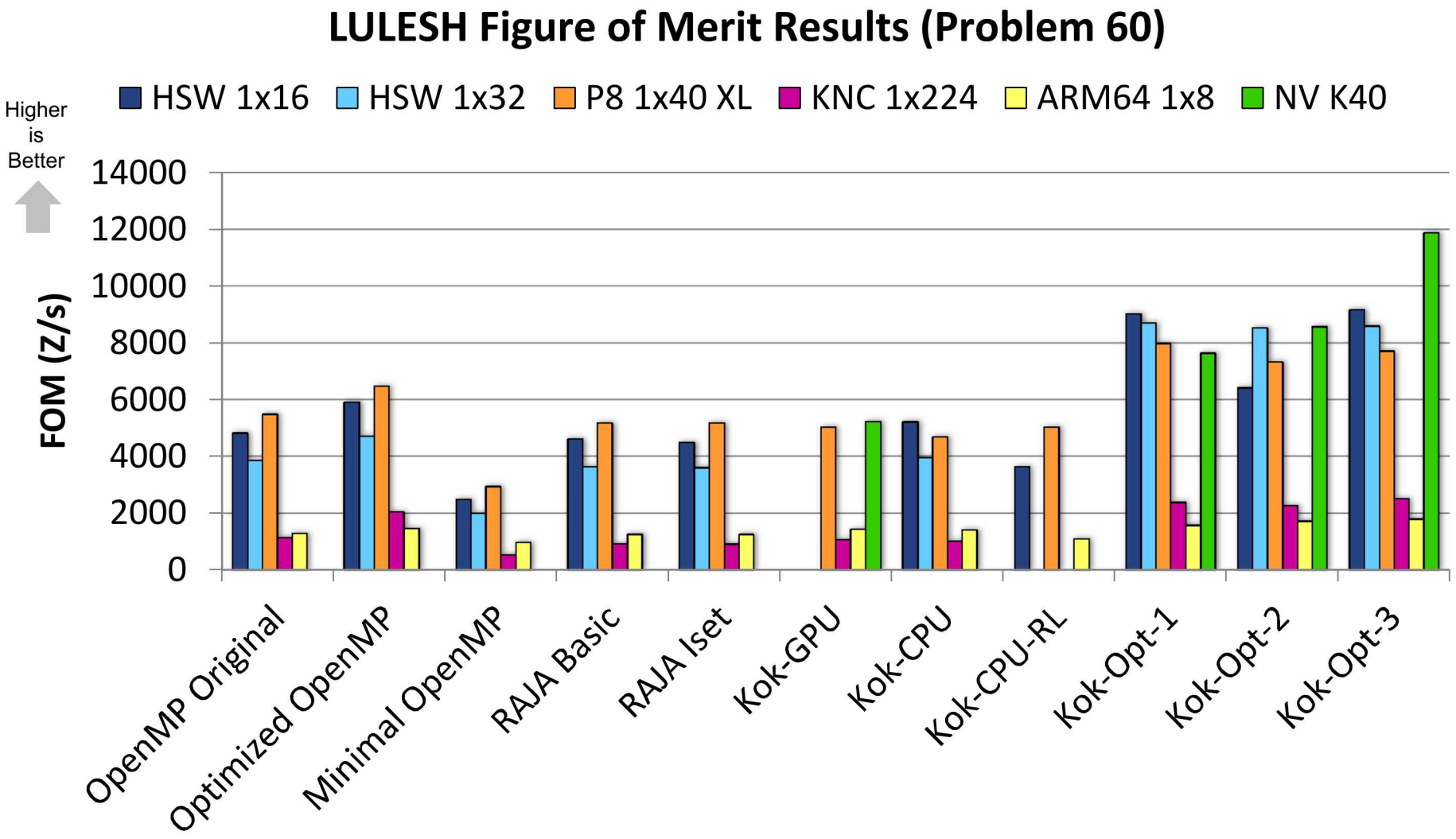
- All configurations used optimized (per platform) MPI process pinning, thread affinities and job configurations
  - Lots of research at Sandia using Mantevo over last four years to understand these issues
  - An on-going process but can give >2X performance difference

Performance Portability Metrics



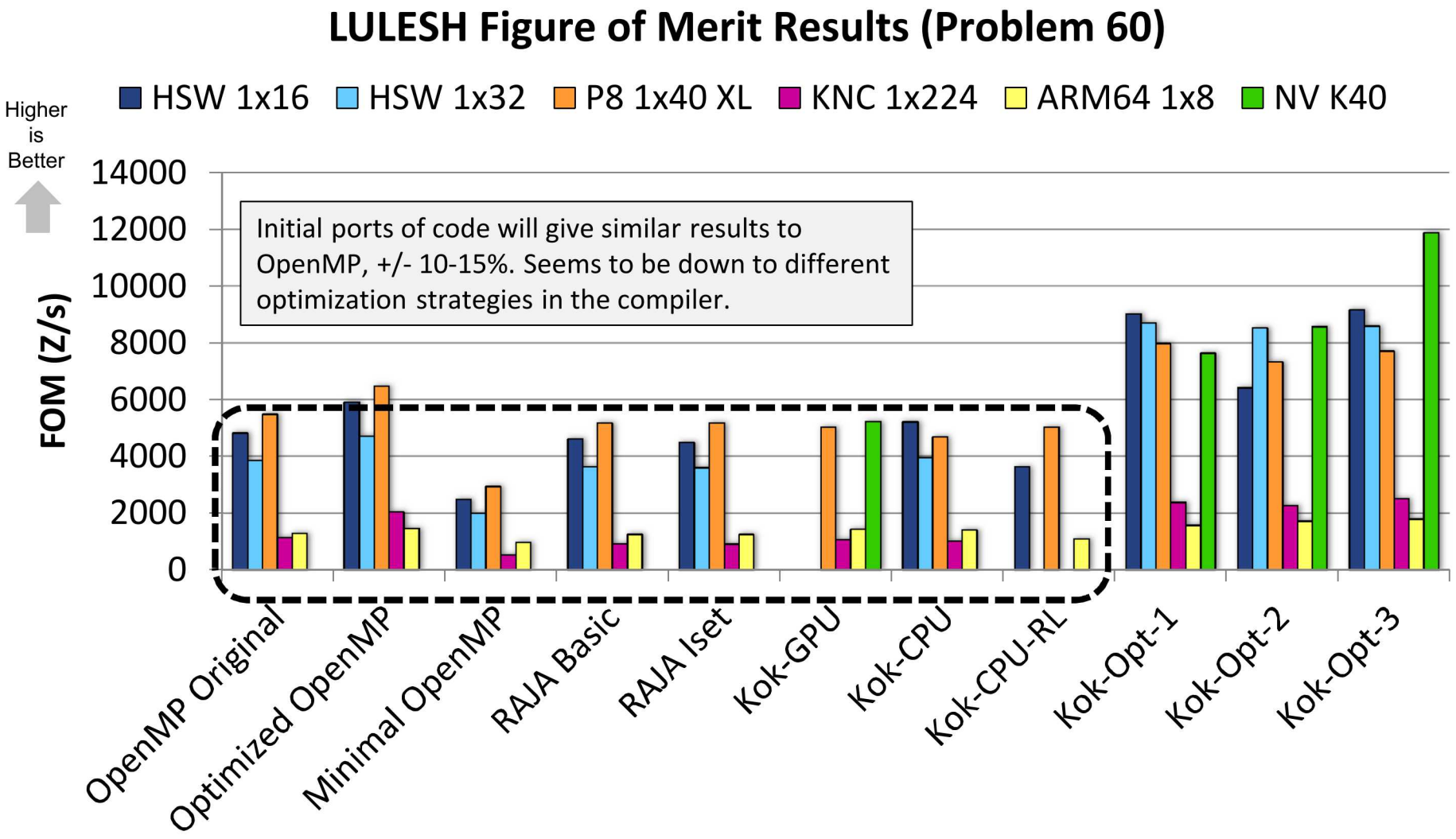
Results by Dennis Dinge, Christian Trott and Si Hammond

Performance Portability Metrics

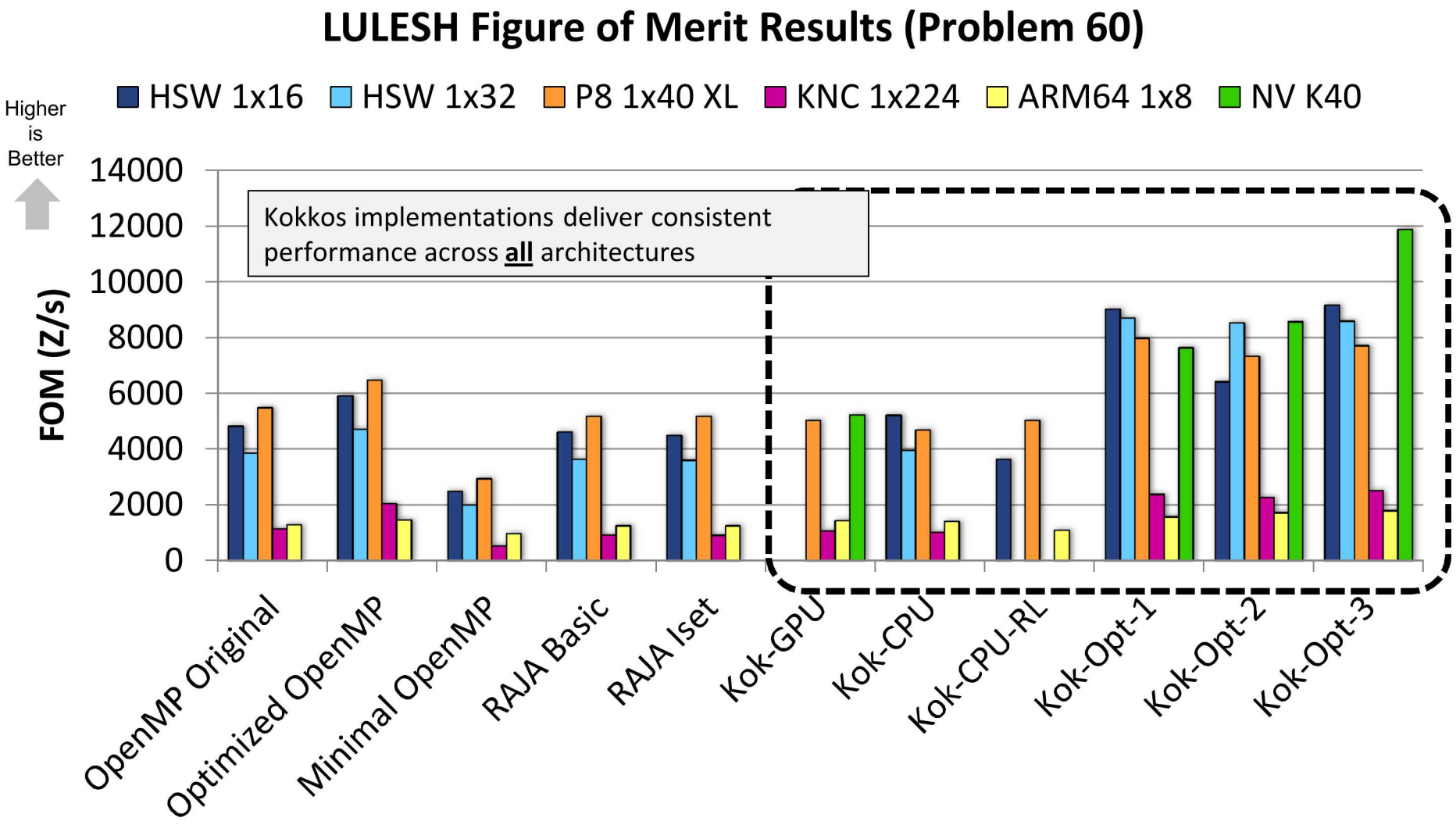


Results by Dennis Dinge, Christian Trott and Si Hammond

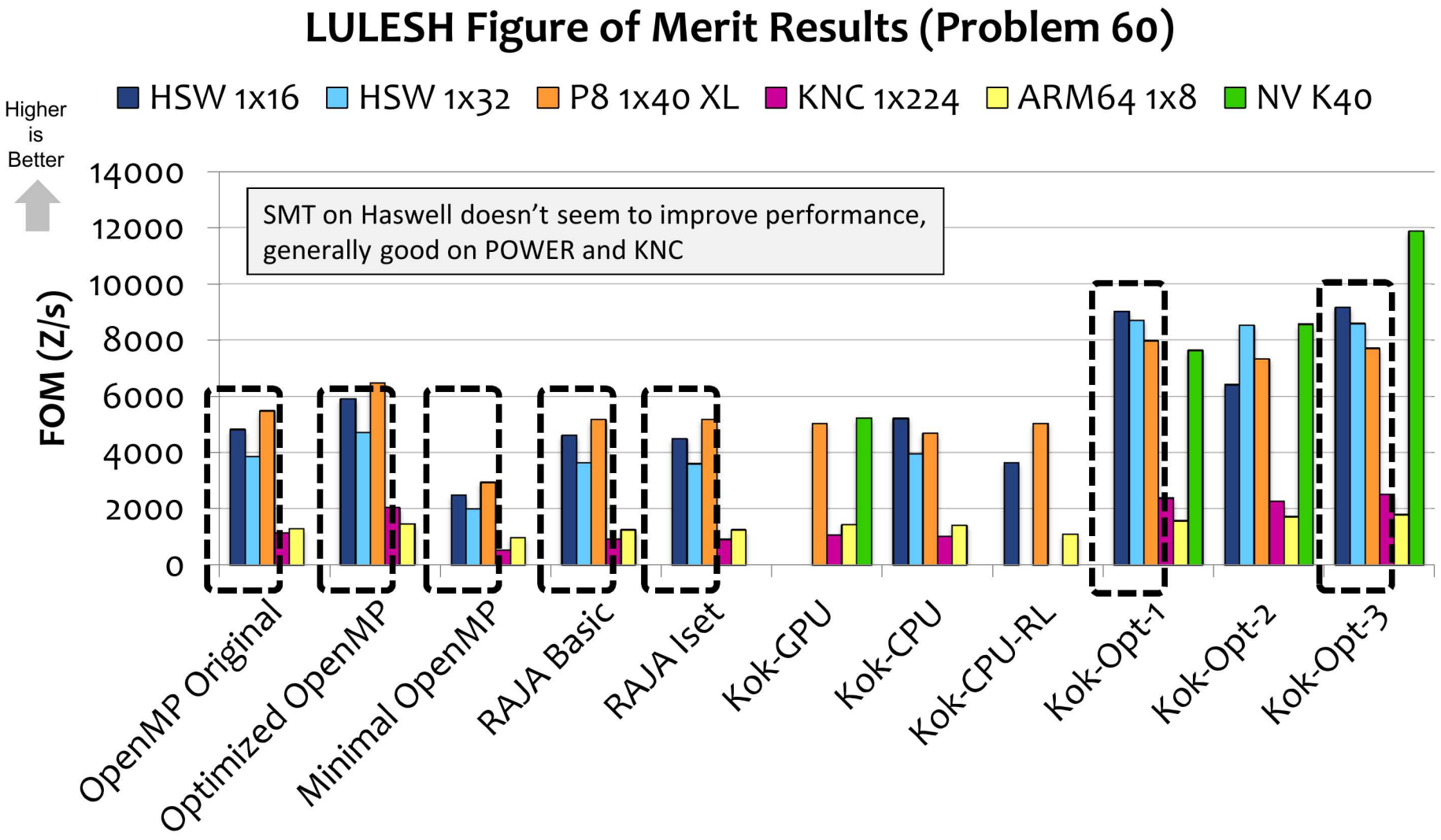
Performance Portability Metrics



Performance Portability Metrics



Performance Portability Metrics



Results by Dennis Dinge, Christian Trott and Si Hammond

Thoughts and Experiences

- These problem sizes are small relative to some of the systems
  - O(100) – O(200) MB in problem size
  - POWER8 – very large memory, large caches (particularly L4)
  - GPU – needs more parallelism

- We are trying to capture performance effects based on feedback from LULESH developers
  - But larger problems help our optimizations even more

- Not necessarily demonstrating the best potential FOM performance
  - Can get up to 2X these FOM figures from our implementations

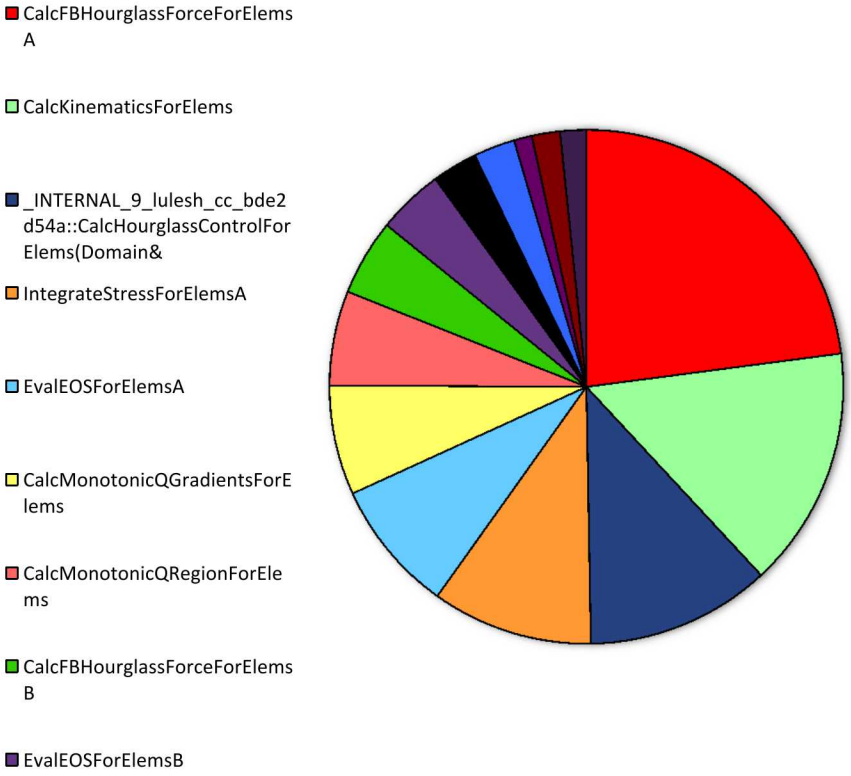
Kernel Analysis for Kokkos Applications

- **Consistent profiling across architectures is hard**
  - Vtune does not like to profile deep in OpenMP hierarchies which are enclosed in headers
  - Nsight manages OK
  - Not clear that tools understand C++ abstraction layers

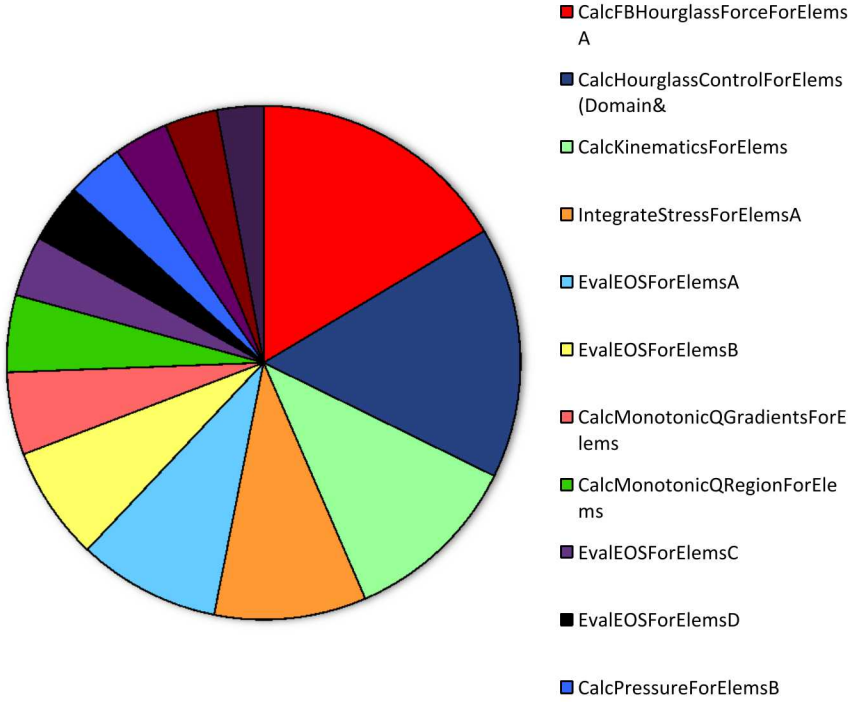
- **KokkosP Profiling Layer**
  - Recent addition to Kokkos, option to always compile in
  - Tools dynamically loaded, can be stacked, lightweight
  - Expose calling structure of kernels and devices to profiler
  - Better context awareness of what execution is being requested
  - Still very early prototype but shows some promise

KokkosP Kernel Comparison of Kokkos Opt 1

Haswell 1x16 S=45 I=1000



POWER8 1x40 S=45 I=1000



See similar breakdown across architectures but we can profile them all using one tool

Sandia National Laboratories

Evaluating Effort to Develop Versions using
Performance Portable C++ Abstraction Layers

PROGRAMMER PRODUCTIVITY OF LULESH VERSIONS

How do we calculate “productivity”?

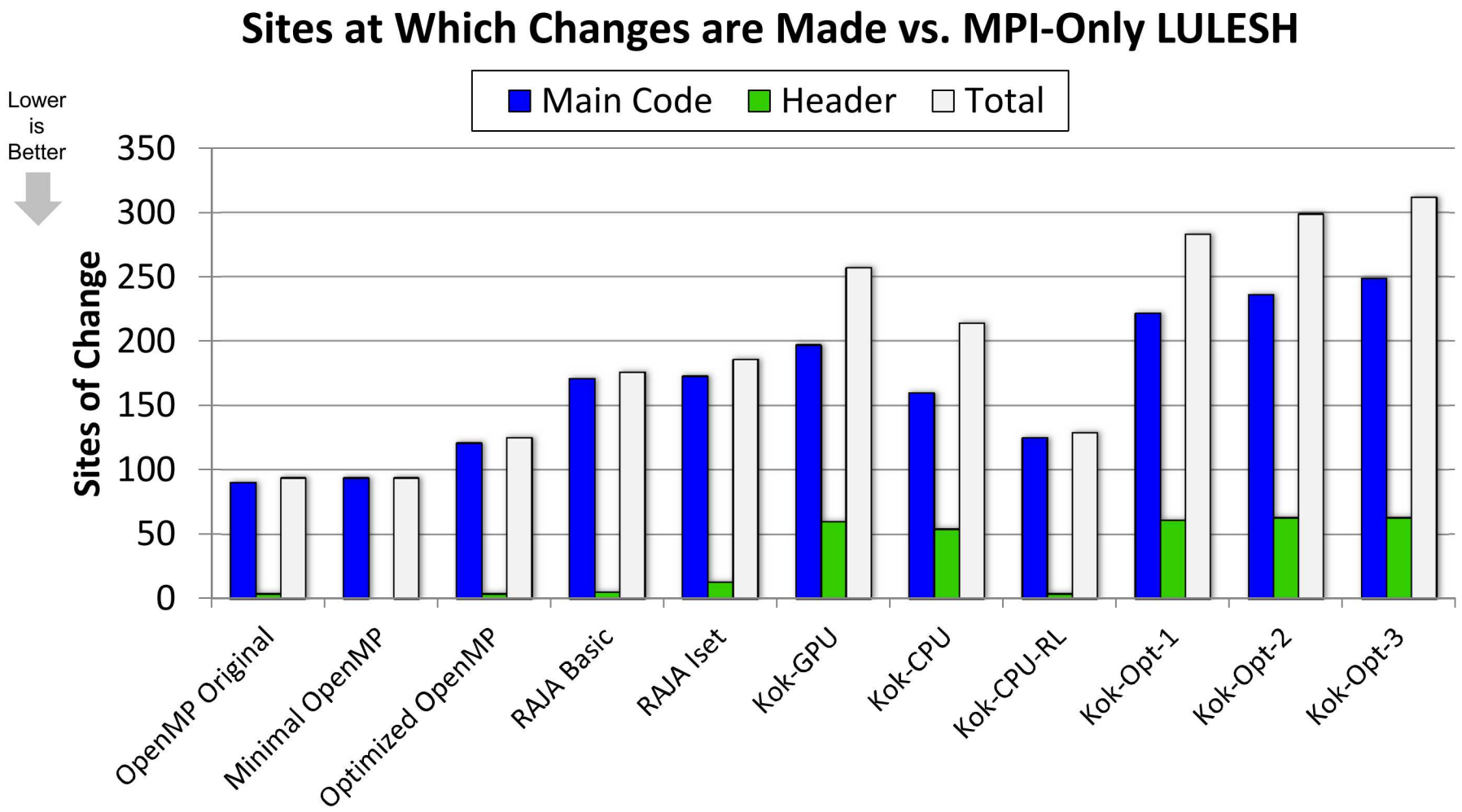
- With great difficulty – lots of discussion in the community about what this *really* means

- Our approach:
  1. Remove all comments from the code
  2. Utilize the clang-format LLVM tool with “Google” code option
  3. Compare the number of sites using Apple’s FileMerge tool
  4. Compare the lines added/removed using diff –b –w <paths>

- Not perfect and we have hand modified code of *all* versions to bring the counts more into line (and to be fair wherever possible)
- Point is to show approximate level of programmer effort not be precisely quantitative because coding style largely down to individual

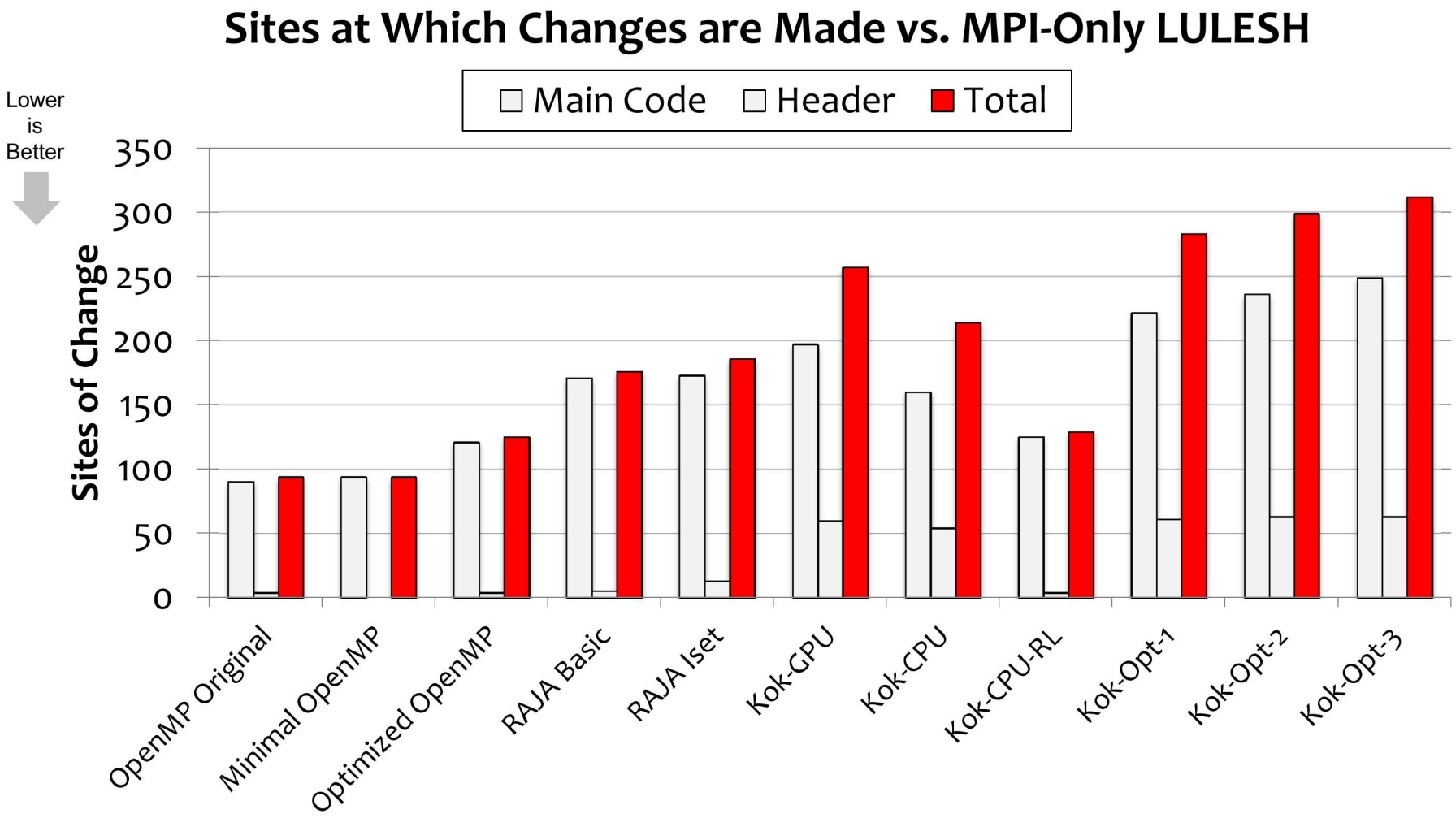
http://clang.llvm.org/docs/ClangFormat.html

Count of Sites at Which Changes are Made



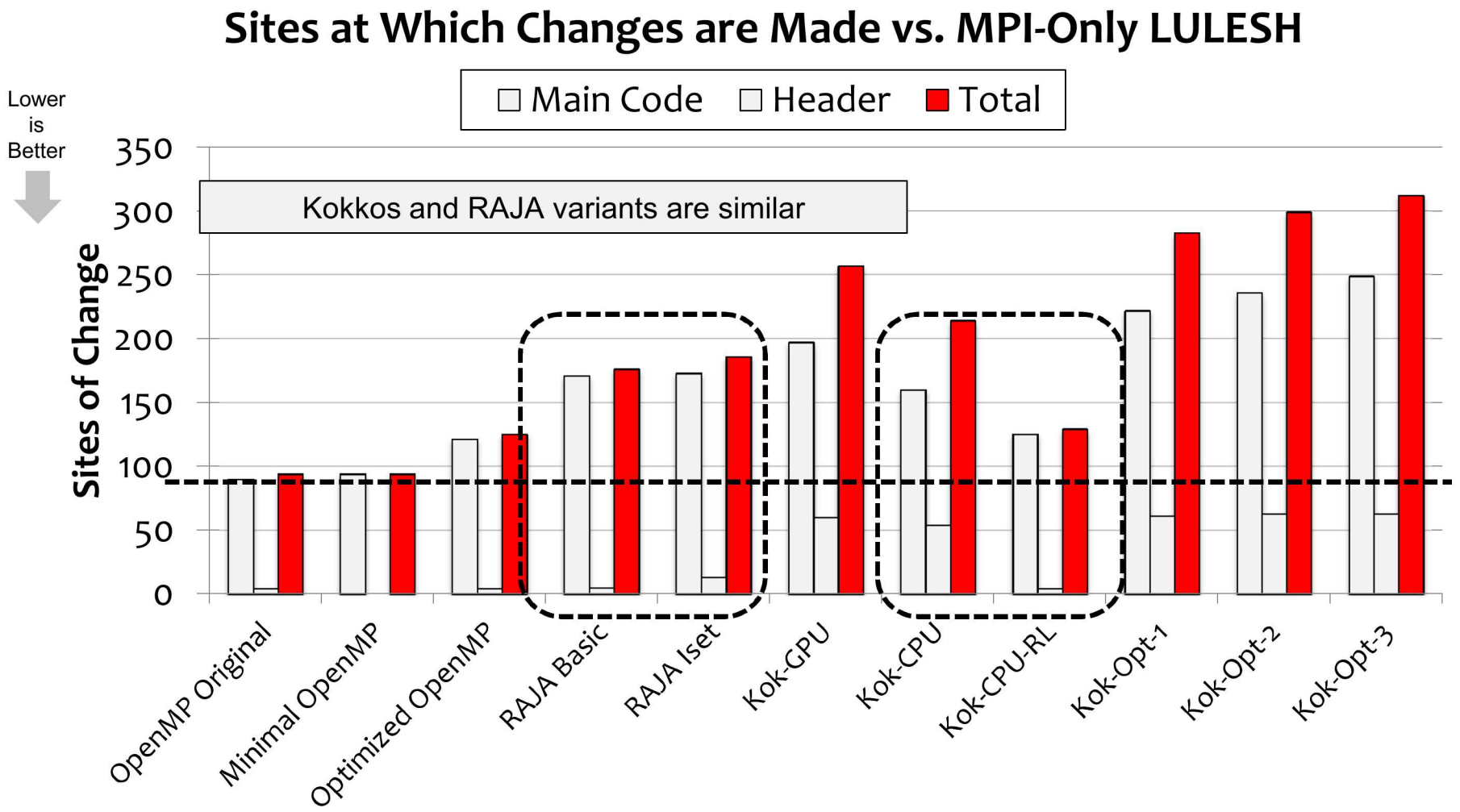
Results by Dennis Dinge, Christian Trott and Si Hammond

Count of Sites at Which Changes are Made



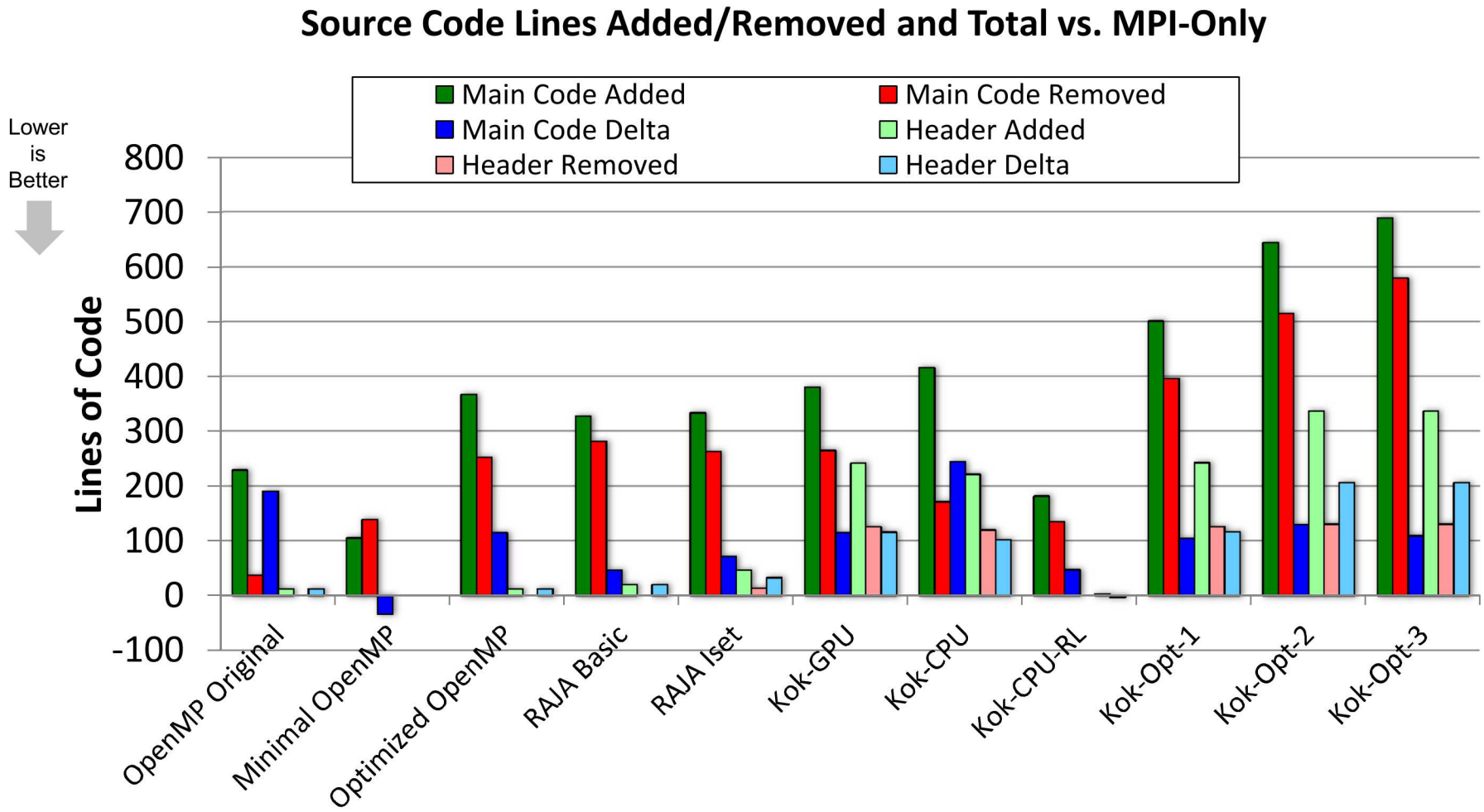
Results by Dennis Dinge, Christian Trott and Si Hammond

Count of Sites at Which Changes are Made



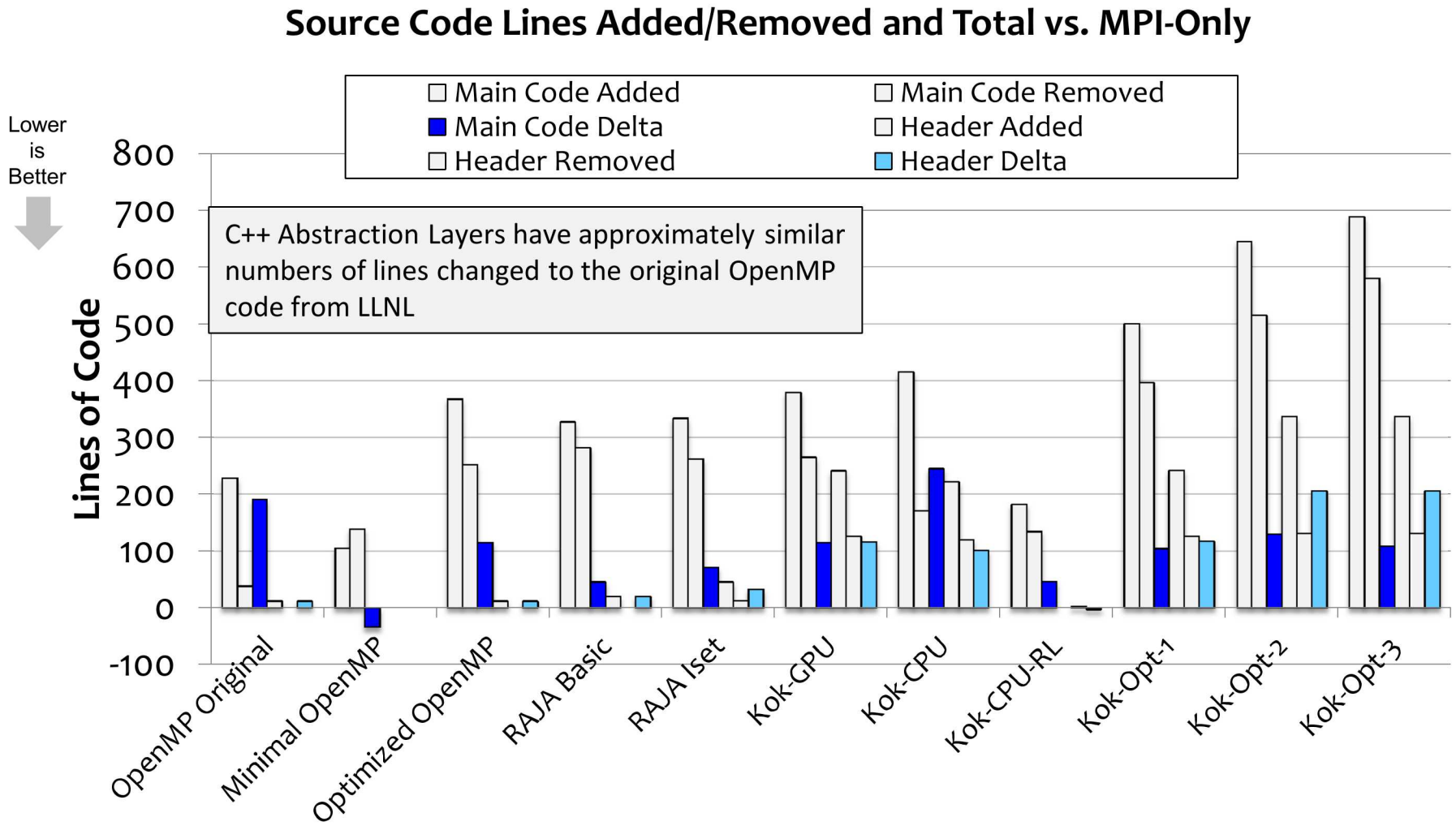
Results by Dennis Dinge, Christian Trott and Si Hammond

Source Code Line Changes

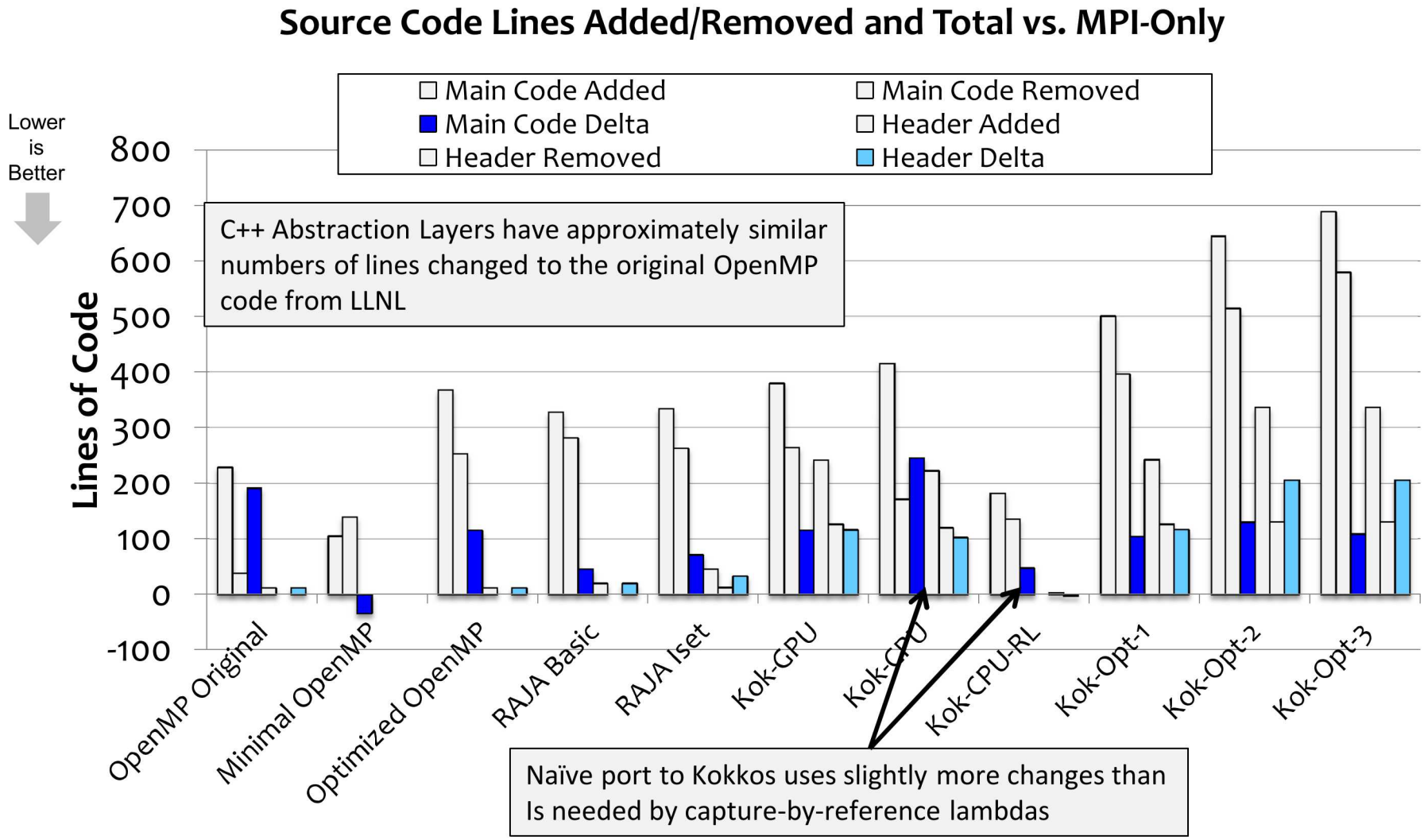


Results by Dennis Dinge, Christian Trott and Si Hammond

Source Code Line Changes



Source Code Line Changes



Results by Dennis Dinge, Christian Trott and Si Hammond

Programmer Development Time

- Initial Kokkos-CPU port by Dennis took a few months
  - No threading/OpenMP/Kokkos experience for code development
  - Lots of correctness and performance issues came up
  - Initial experience with programmer tools and profilers
- Kokkos optimized implementations
  - O(few weeks) of Christian's time ("Kokkos-expert")
- OpenMP initial and optimized implementations
  - O(few days - week) of Si's time written on a plane
- These are not significant amounts of FTE but the code is small in comparison to production settings (but code groups are larger and better resourced)
- Difficult (impossible?) to do a deep quantitative comparison

What can we take away?

- C++ abstraction layers are using similar numbers of changes in code (both code sites and SLOC-delta) to directives

- Perhaps to be expected given implementation strategy is similar in unoptimized variants of the code
  - This is a good thing for developers – hard work is in developing the parallel algorithm, not in how it is expressed in source code

- Looking at changing roughly 15% of the code to get initial parallel versions in this example
  - Warning: example is friendly to parallelism because of its heritage

- Do we need directives in application code at all?

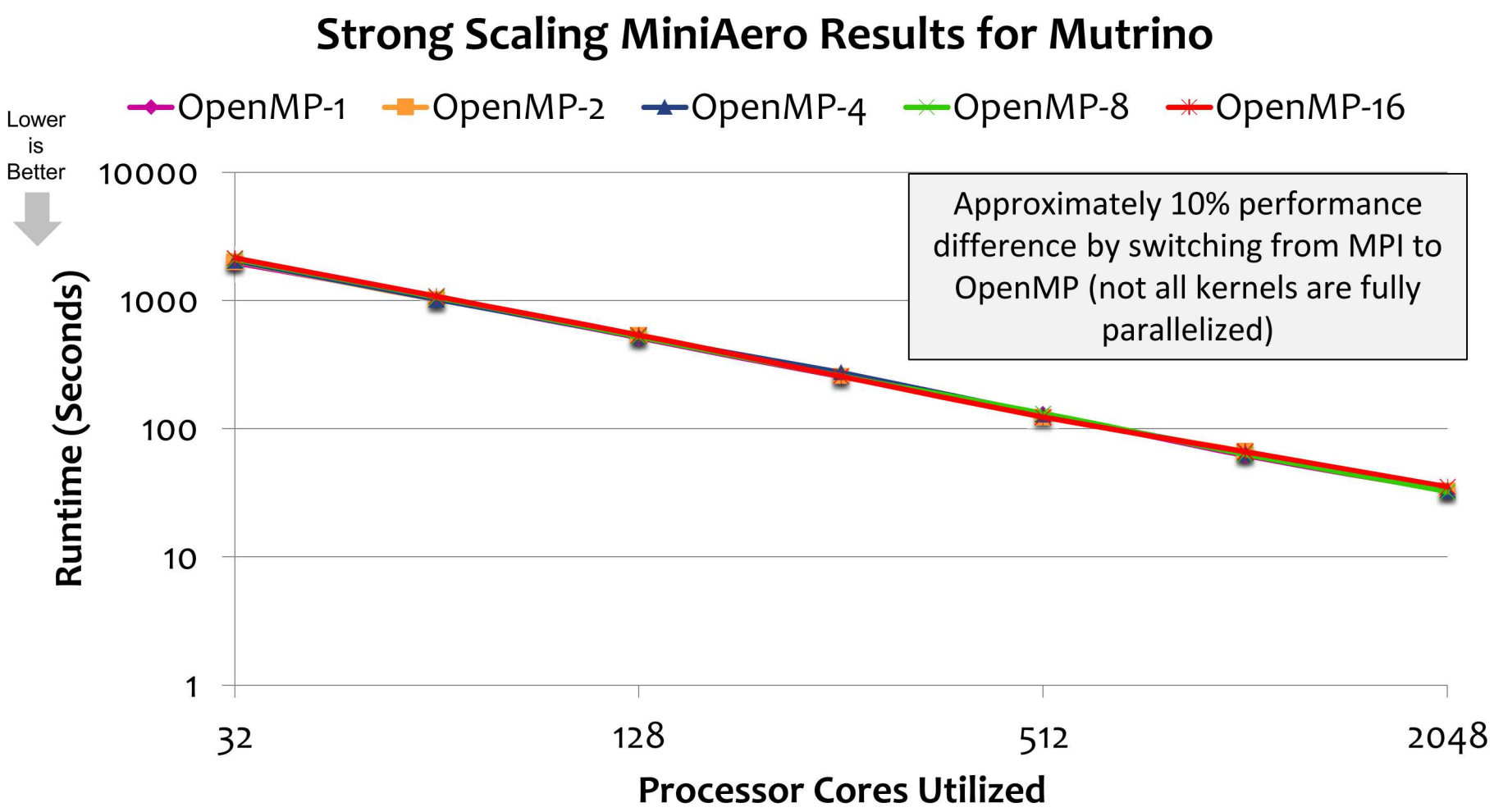
Sandia National Laboratories

ANALYSIS OF MINIAERO

MiniAero Overview

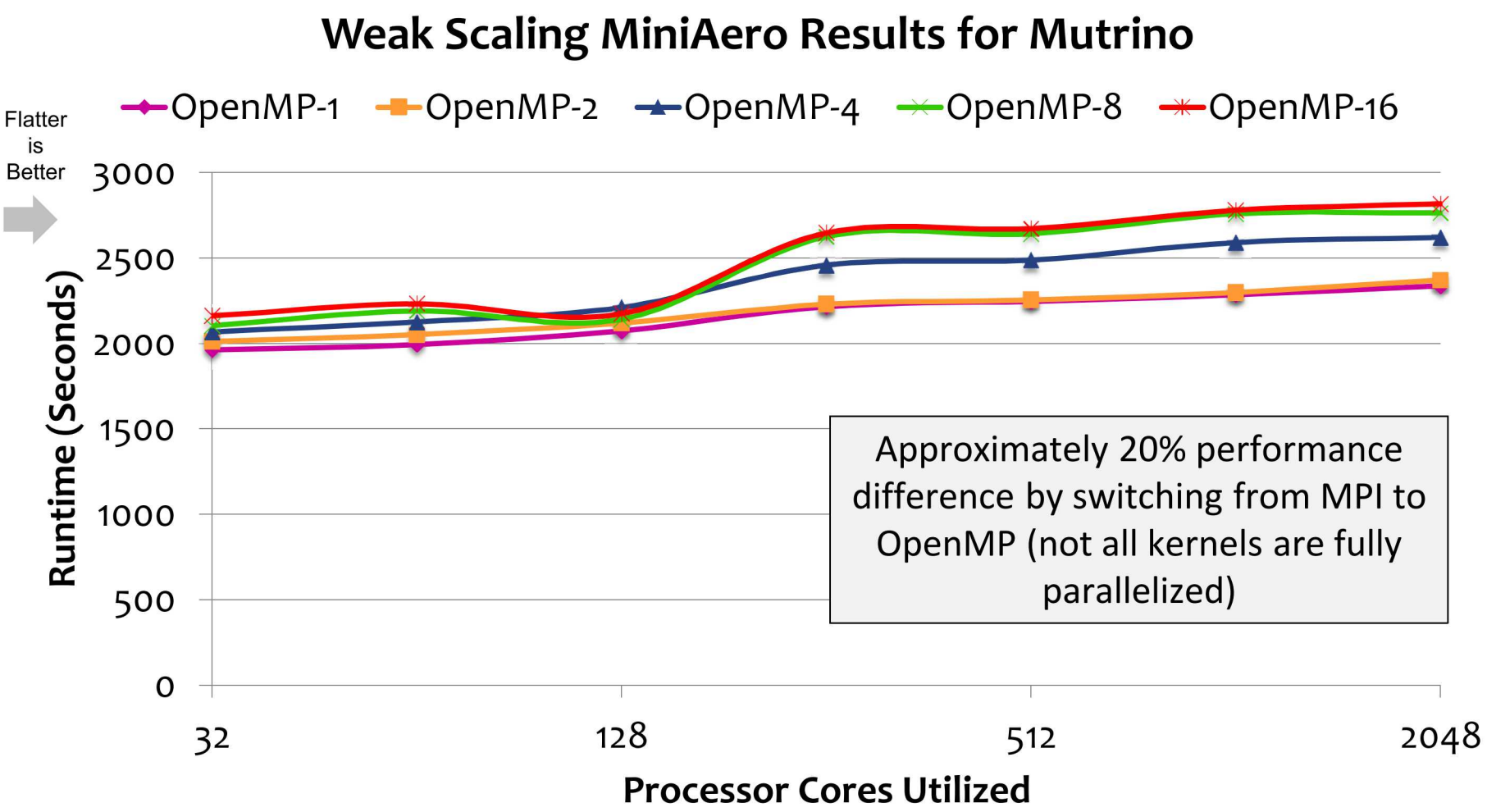
- Originally written by Ken Franko (now at Google)
  - Added to Mantevo suite in 2014
- Designed for exploration of Kokkos programming model
  - Not to be used as a proxy for production algorithms
  - Did not have an “original” OpenMP or serial implementation
- Different options for threaded algorithm to aggregate values onto the mesh
  - Use of atomics operations
  - Use of gather/sum

MiniAero Scaling Analysis on Trinity Test Machines



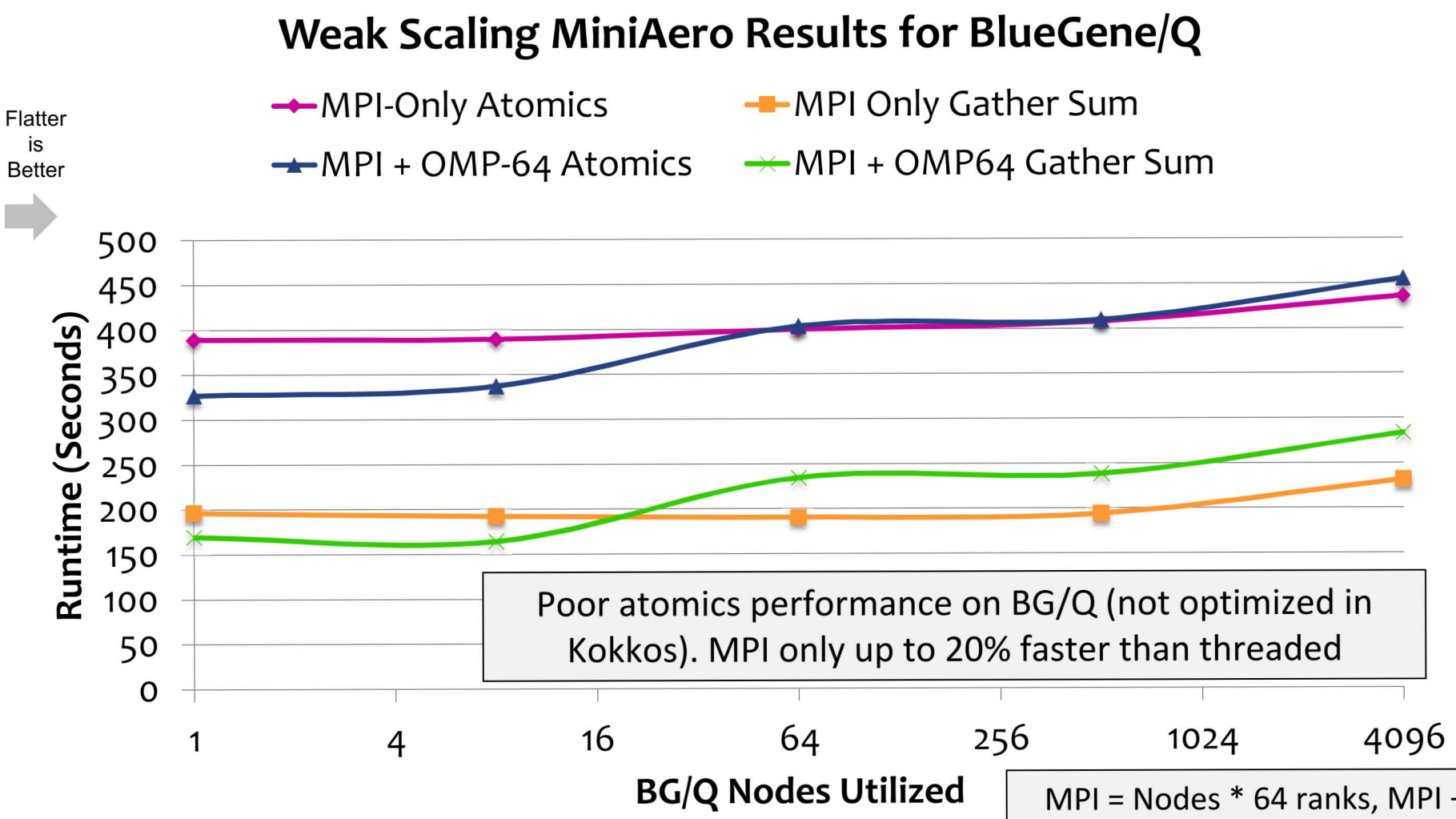
Results by Jeanine Cook and Courtenay Vaughan

MiniAero Scaling Analysis on Trinity Test Machines

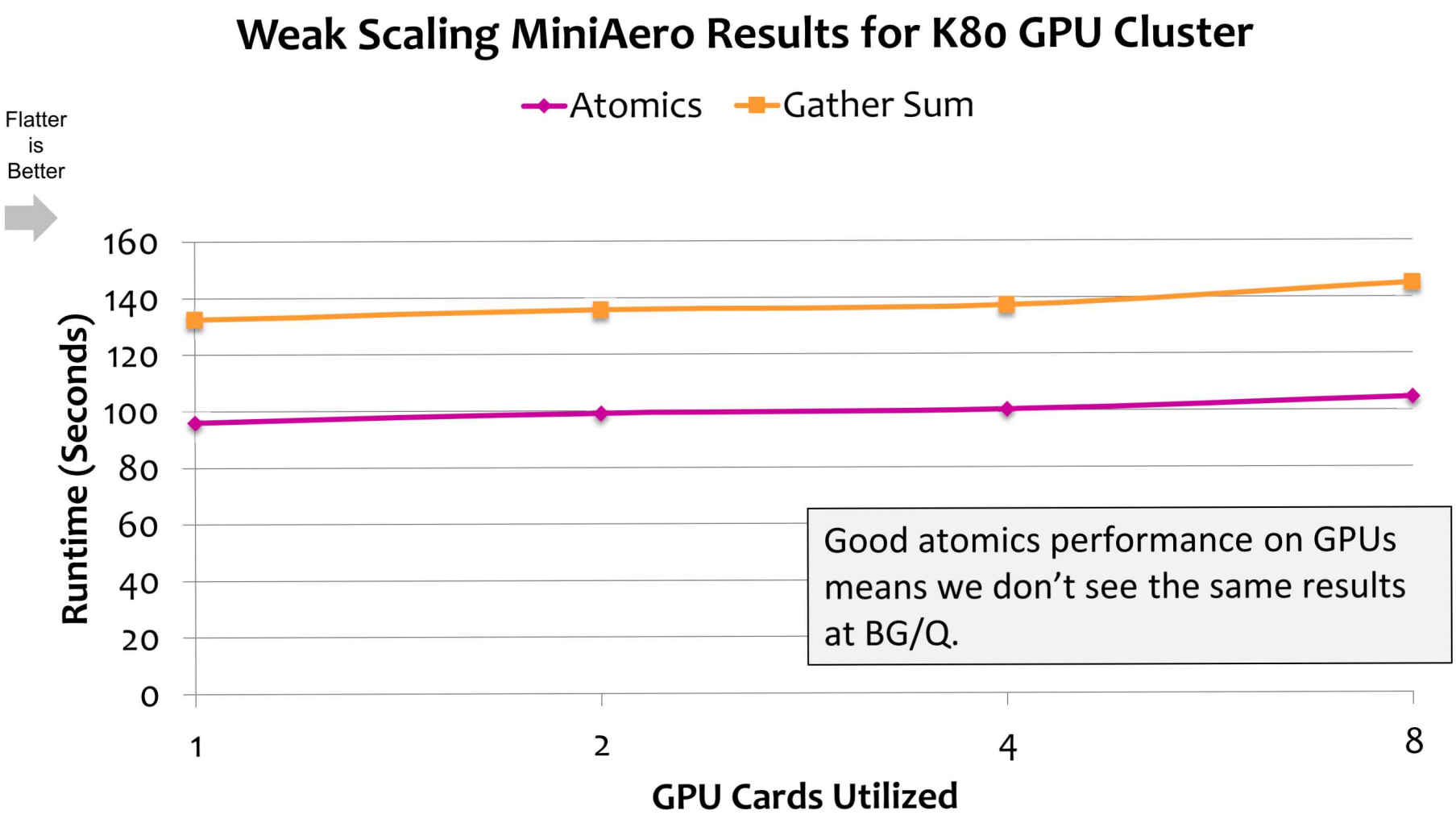


Results by Jeanine Cook and Courtenay Vaughan

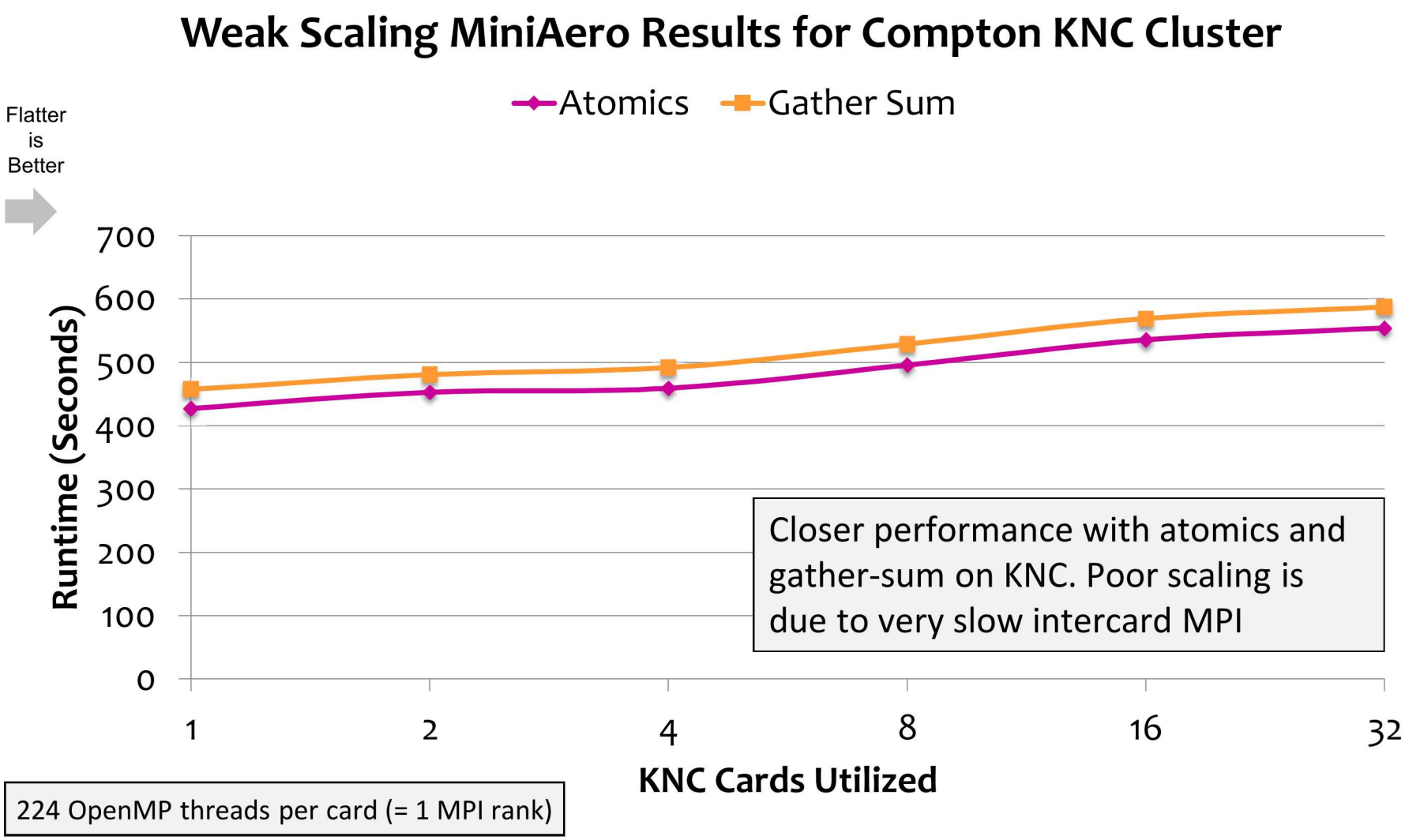
MiniAero Scaling Analysis on BlueGene/Q



MiniAero Scaling on GPU Clusters

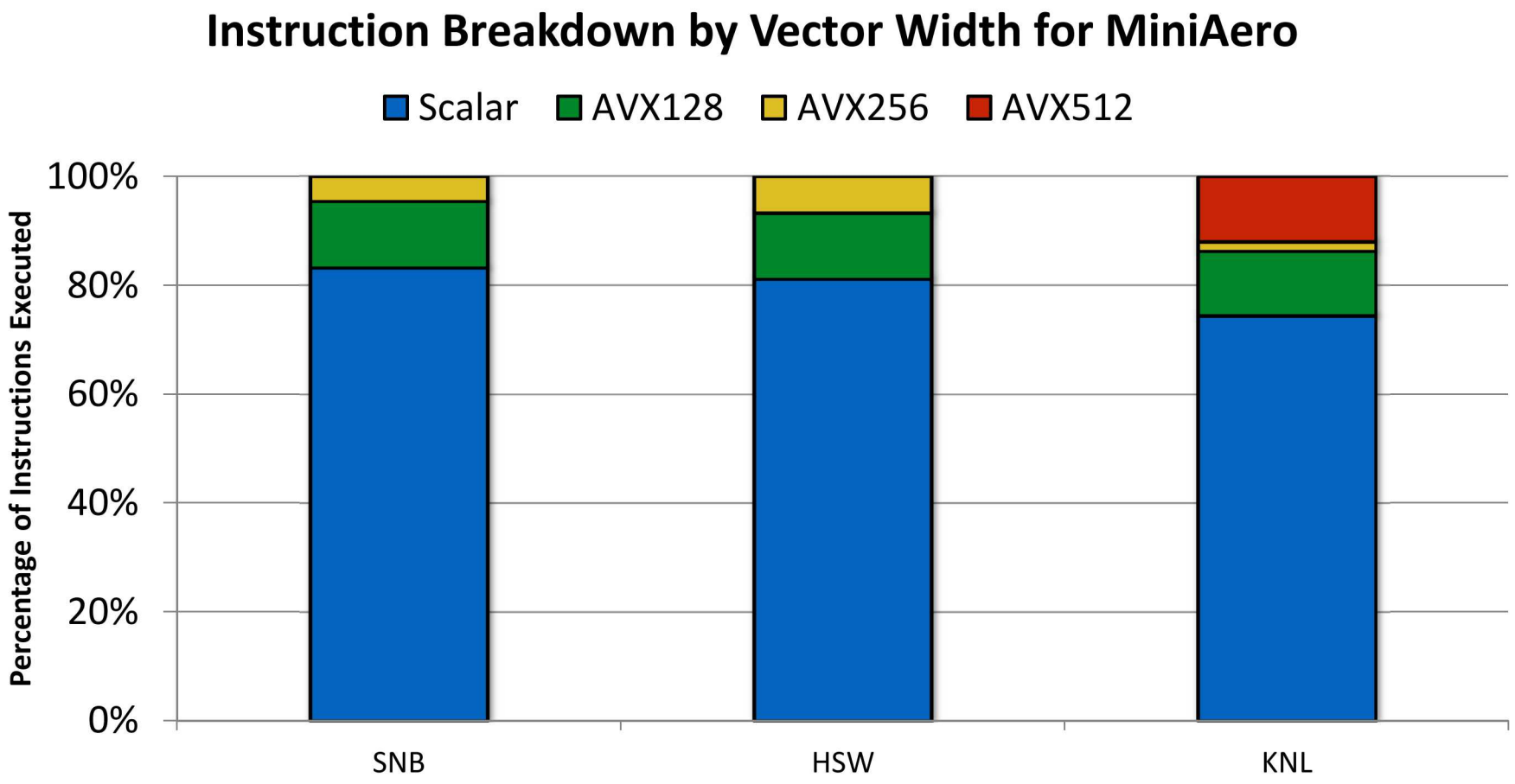


MiniAero Scaling on KNC Clusters



224 OpenMP threads per card (= 1 MPI rank)

Emulation and Instruction Analysis for KNL



- Covers all instructions executed (dynamic stream) including move operations and register clears

MiniAero Summary

- Question as to whether **exactly** the same algorithm will run on **all** architectures well – atomics vs. gather-scatter

- Open question which requires further research

- May not be able to find a single source which always runs truly well everywhere
  - Is not intrinsic to Kokkos, the same issue is true for OpenMP, RAJA *etc*

- Continues to reinforce why we need codesign and research into our code performance

- Clearly still need to look at poor vectorization levels for Trinity machines

CONCLUSIONS AND DISCUSSION

Summary

- **Showed performance and portability of two Kokkos mini-app implementations across ASC Advanced Architecture Test Beds**

- Strong performance across architectures for LULESH
  - Often as strong or stronger than equivalent OpenMP code
- Initial expectations for use of Haswell, POWER and GPU systems
  - Knights Landing still remains an unknown due to significant changes over Knights Corner cards
- Evaluated programmer productivity for LULESH
  - C++ abstraction layers are approximately equivalent to well optimized OpenMP code in sites of code change and number of source lines

Feedback to Vendors/Community

- Kokkos is now on github.com (fully open source and free for everyone)
  - Full public release of the most up to date development branches
  - Strong engagement with NVIDIA, AMD and IBM, initial engagement with Intel
  - Feedback to IBM and Cray on compiler issues, during this L2 both now compile miniapps successfully
  - Now has initial support for Knights Landing compile path
- Implementations using Kokkos will be available for the community in Mantevo release for SC15
- Poster submitted to SC15 covering OpenMP and Kokkos studies (no RAJA)
- Clearly still a need in some areas for better optimization support in compilers
  - See very varied inlining, optimization, vectorization etc. More time and more focus by the labs will help
- Committed to C++ abstraction layer support in development of ATS3 RFP

Productivity

- Productivity in Kokkos in some ways has always been behind portability and performance
  - We needed to learn the best approach before we could work out how to enhance programmer productivity
- Have learned a lot through discussions with RAJA team on why this is important and through our own application work on LAMMPS, Trilinos, Albany, SIERRA *etc*
- Have a much stronger story in productivity on the parallel execution/dispatch
  - This codesign study has helped inform us further
- Kokkos has strong story for data management
  - Initial work on efficient parallel STL-like containers

- Our experience is 90% of the work is in making the algorithm parallel and optimizing the data structures not in the specific way its written

Kokkos in the Community

- Published a Kokkos Programming Guide in 2015
  - Based on lots of feedback from community
  - Covers general concepts and themes of Kokkos
- Kokkos Training Material
  - 200 tutorial slide deck
  - Multiple examples with varying levels of complexity
- Kokkos Tutorial at Sandia in September
  - Over 80 registered attendees
  - Will work on multi-core, many-core and GPU Sandia test beds
- Tutorial at ACM/IEEE Supercomputing in November 2015

Acknowledgments

- Application Performance Team at Sandia
  - Dave Resnick, Jim Thomkins, Sue Phelps
- ASC Advanced Architecture Test Beds at Sandia
  - Project Management and System Administration Team
  - Jim Brandt, Ann Gentile, Victor Kuhns, Nate Gauntt, Jason Repik, T.J. Lee, Jim Laros, Sue Kelly
- SIERRA Code Teams for inputs (-SM, -SD and -TF)
  - Mike Glass, Mike Tupek, Kendall Pierson, Nate Crane, Mark Mereweather, Travis Fisher & many others
- Kokkos Development Team
  - Carter Edwards, Mark Hoemmen, Dan Sunderland, Irina Dimenshenko & others
- ASC L2 Review Committee
- Jeff Keasler, Ian Karlin and Rich Hornung (LLNL) for inputs on RAJA, LULESH and general programming model discussion
  - We have learned a great deal from you folks

Sandia National Laboratories

BACKUP SLIDES

MiniAero Thread Scaling on Cray XC30