NNSA Investigation of Advanced Programming Models and Runtime Systems for Exascale

Patrick McCormick
Programming Models Team Lead / ASC Project Lead
Los Alamos National Laboratory

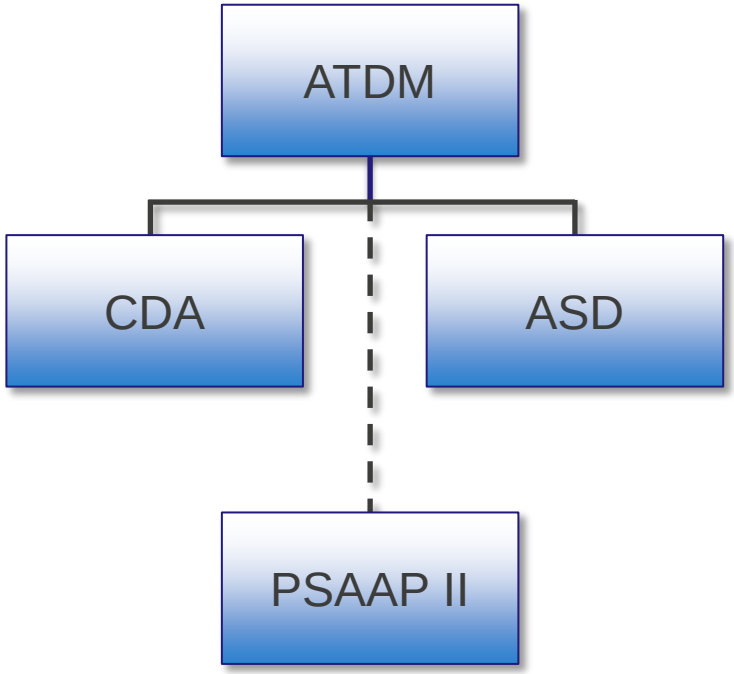
April 4, 2016

ASCR Advisory Committee Meeting
American Geophysical Union
Washington, DC

Operated by Los Alamos National Security, LLC for the U.S. Department of Energy's NNSA

Addressing Disruptive Changes: Formation of the *Advanced Technology Development and Mitigation* Subprogram

- **ATDM formed in FY14.**
- **Split into two primary components:**
  - Next generation Code Development and Applications (CDA)
  - Next generation Architecture and Software Development (ASD)
- **Foundation for the NNSA's execution of ECI responsibilities.**
  - Engaged with ASCR to address the barriers to exascale and evolving architectures (including *Forward activities)
- **Supported by phase 2 of the Predictive Science Academic Alliance Program (PSAAP)**



University of Florida
University of Illinois
University of Notre Dame
Stanford University
Texas A&M University
University of Utah

The Balancing Act of Meeting Requirements for Programming Environments within ATDM

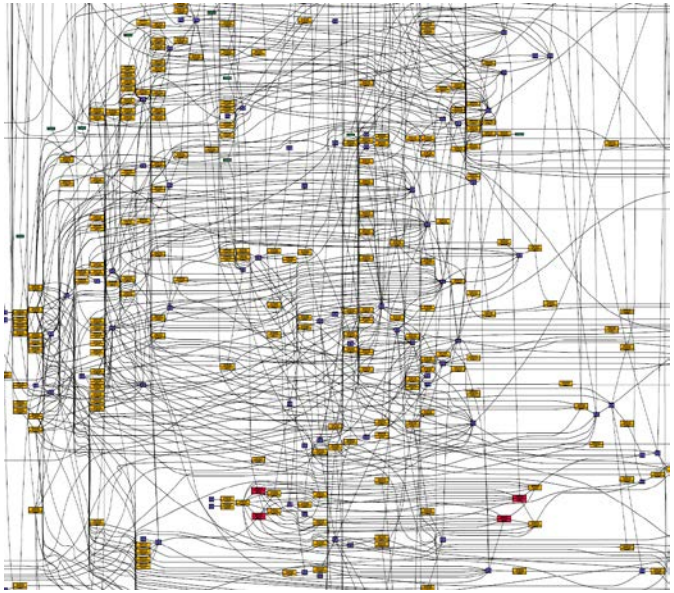
- **Achieve a *balance* between:**
  - Performance
  - Portability
  - Productivity/Programmability
  - Support for multiple languages and models (interoperability)
  - Don’t forget: correctness and reproducibility

- **ATDM direction: *Consider moving beyond the status quo for addressing these goals***
  - Explore new models / more effective abstractions
  - Shape and influence standards
  - Tight coupling between CDA and ASD activities

The Importance of Programming Abstractions for Achieving Our Goals

Today: Imperative, explicit data movement:

- ***Focus on control flow, explicit parallelism and low-level data abstraction.***



- *How much work should I do?*
- *Is this performance portable?*
- *When does forward progress really occur?*
- *What if I have more work and data movement happening in DoWork?*
  - *What resources are in use? Where is the data? Who is using it and how?*
- *Is this modular?*

Based on Mike Bauer's Thesis (Stanford), *Legion: Programming Distributed Heterogeneous Architectures with Logical Regions*

Outline: Layers of Abstraction from the Node Outward

- **Node Level Models**
  - Kokkos (Sandia) and RAJA (Livermore)
    - C++ template metaprogramming for performance portability
    - *Functional-like* programming techniques (e.g. functors/labmdas)

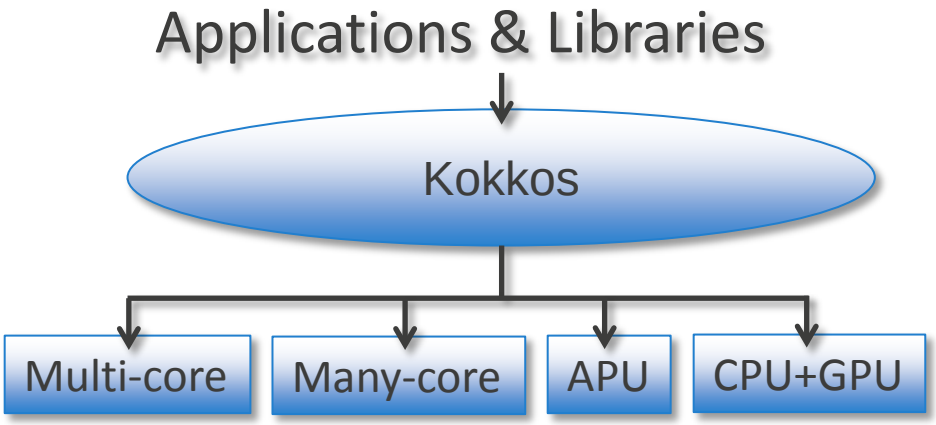
- **System Level Models**
  - Legion (LANL)
  - DARMA (Sandia)

- **Conclusions**

Kokkos: Performance Portable Thread-Parallel Programming Model

Sandia National Laboratories

- **Open source, C++11-based library for node-level programming: application identifies parallelizable grains of *computations and data***



- ✓ **Multicore CPU** - including NUMA architectural concerns
- ✓ **Intel Xeon Phi (KNC)** – testbed prototype toward Trinity / ATS-1
- ✓ **NVIDIA GPU (Kepler)** – testbed prototype toward Sierra / ATS-2
- ✧ **IBM Power 8** – testbed prototype toward Sierra / ATS-2
- ✧ **AMD Fusion** – via collaboration with AMD
  - ✓ Regularly and extensively tested
  - ✧ Ramping up testing

https://github.com/kokkos/kokkos

Kokkos Abstractions: Patterns, Policies, and Spaces

Sandia National Laboratories

- **Parallel Pattern of user's computations**
  - parallel_for, parallel_reduce, parallel_scan, task-graph, ... (*extensible*)
- **Execution Policy tells how the computations will be executed**
  - Static scheduling, dynamic scheduling, thread-teams, ... (*extensible*)
- **Execution Space tells where the computations will execute**
  - Which cores, numa region, GPU, ... (*extensible*)
- **Memory Space tells where user data resides**
  - Host memory, GPU memory, high bandwidth memory, ... (*extensible*)
- **Layout (policy) tells how user data is laid out in memory**
  - Row-major, column-major, array-of-struct, struct-of-array … (*extensible*)

```
parallel_for( nrow, KOKKOS_LAMBDA( int i ){
  for ( int j = irow[i] ; j < irow[i+1] ; ++j )
    y[i] += A[j] * x[ jcol[j] ];
});
```

RAJA: A Systematic Approach to Node-Level Portability and Tuning

Lawrence Livermore National Laboratory

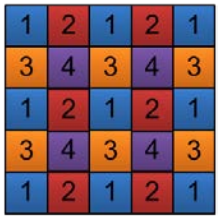
- **Loops are the main conceptual abstraction in RAJA**
  - Based on loop structures and mesh traversal patterns in LLNL ASC codes (many loops O(10K) but only O(10) patterns – RAJA categorizes these patterns.
- **Lightweight, can be adopted incrementally, does not overburden maintenance, allows easy exploration of alternative parallel strategies**
- **Key abstractions:**
  - Traversals & execution policies (loop scheduling, execution, implementation details)
  - IndexSets (iteration space partition, data placement, dependency scheduling)
  - Reduction types (programming model portability)

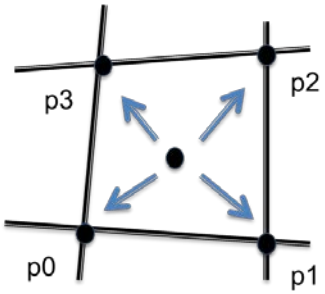
More on RAJA: http://1.usa.gov/1MJXlGd

IndexSets Allow Common Algorithms to Run Safely in Parallel Without Refactoring or Critical Sections

Lawrence Livermore National Laboratory

- **Allows loop traversals to execute groups of work in parallel that guarantee no race conditions in otherwise non thread-safe loops**
- **E.g. Element volume can be distributed to nodes without contention-heavy fine-grained synchronization such as critical sections, atomic operations, and temporary arrays**





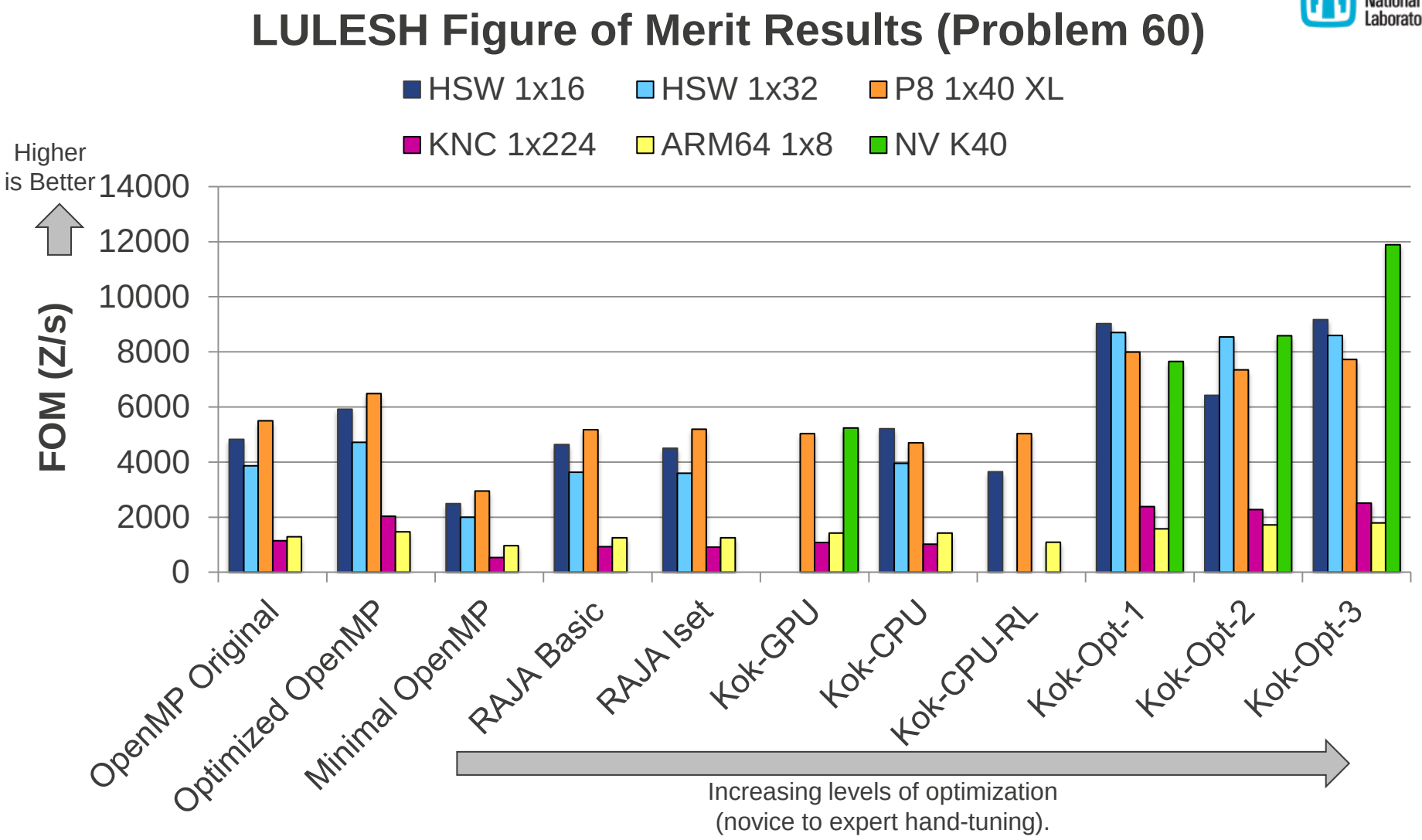

```
forall<colorset>(elemSet, [=] (int elem) {
   int p0 = elemToNodeMap[elem][0];
   int p1 = elemToNodeMap[elem][1];
   int p2 = elemToNodeMap[elem][2];
   int p3 = elemToNodeMap[elem][3];
   double volFrac = elemVol[elem]/4.0 ;
   nodeVol[p0] += volFrac ;
   nodeVol[p1] += volFrac ;
   nodeVol[p2] += volFrac ;
   nodeVol[p3] += volFrac ;
} ) ;
```

Parallel reductions

- **Indexsets allow for ‘contention-light’ coarse-grained locking**

Kokkos Performance Metrics (Summer 2015)

Sandia National Laboratories

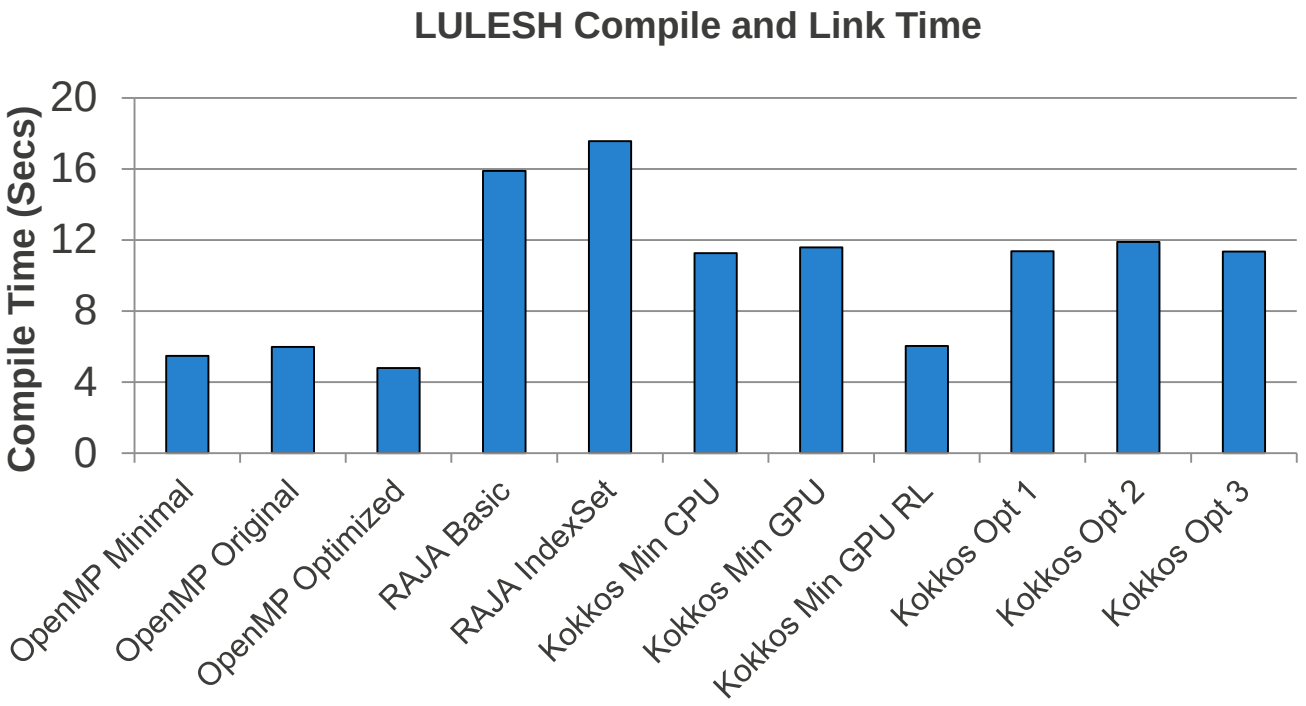


LULESH: https://codesign.llnl.gov/lulesh.php

Results by Dennis Dinge, Christian Trott and Si Hammond

C++ Metaprogramming Impacts on Productivity

- **Template mechanisms can result in:**
  - Long compile times, large executable sizes, code optimization challenges

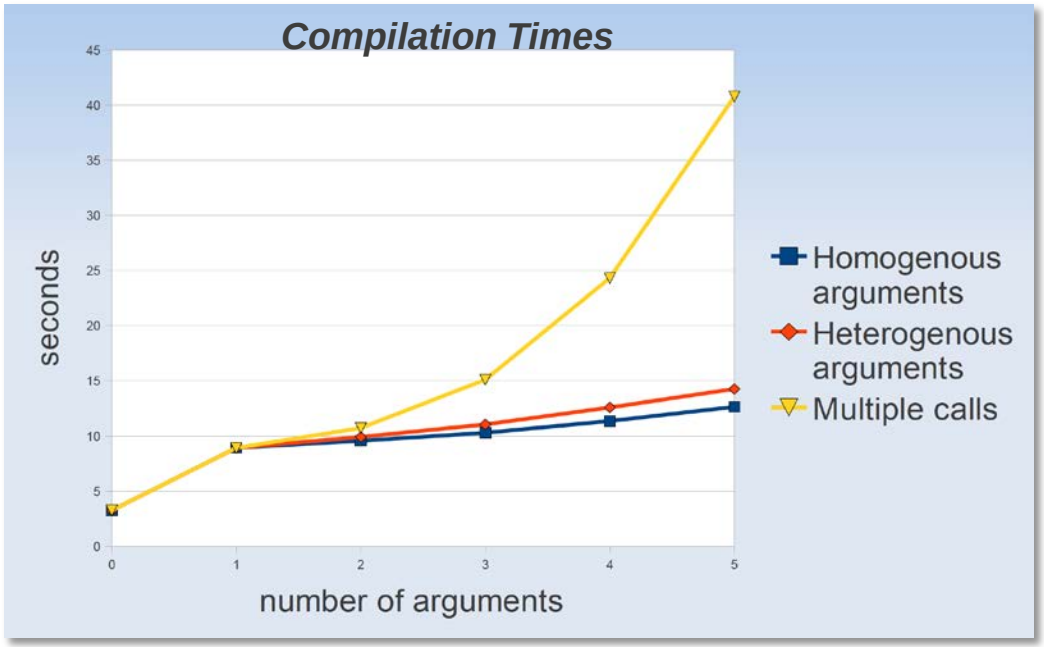


Courtesy: Si Hammond (Sandia)

C++ Metaprogramming Impacts on Productivity

- **Template mechanisms can result in:**
  - Long compile times, large executable sizes, code optimization challenges
  - C++ Embedded DSLs can be even more costly…

```
safe::printf<_S("Hello, %s!")>(”world”);
```

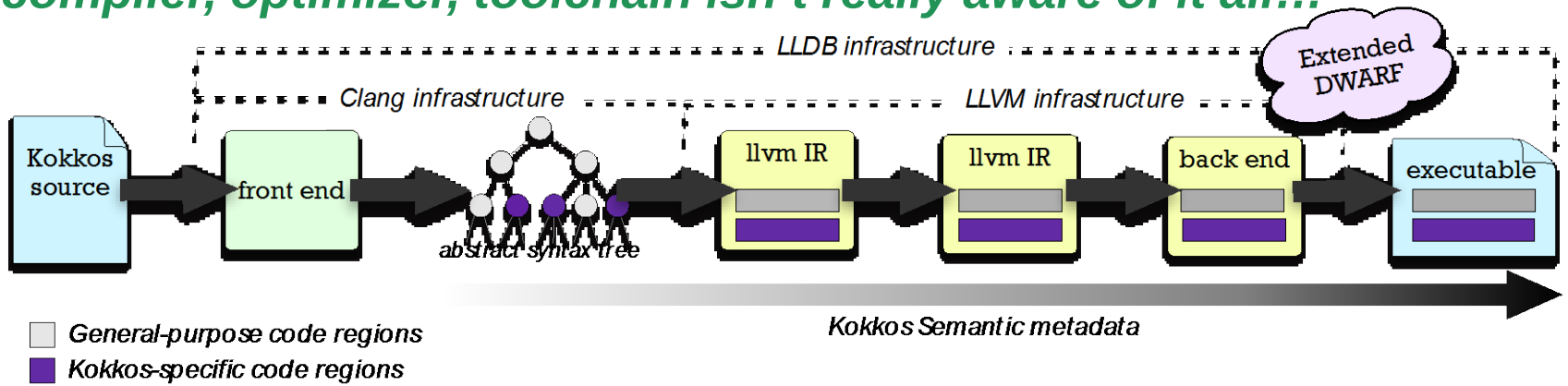


“Domain-specific Language Integration with Compile-time Parser Generator Library”, Zoltan Porkolab and Abel Sinkovics, Proceeding GPCE '10 the Ninth International Conference on Generative Programming and Component Engineering.

The Lack of "Model-Awareness" is a Limitation of C++ Metaprogramming...

IDEAS productivity

We are using a lot of "functional-like" programming but the compiler, optimizer, toolchain isn't really aware of it all…



- **Leveraging LLVM "domain-specific toolchain" approach based on work funded by ASCR (Lucy Nowell) and recent work from the IDEAS Project (ASCR & BER)**
- **Kokkos constructs recognized and skip template expansion and instead use semantics-aware code generation.**
  - SC15 Kokkos tutorial code: Compiles **~4.5x faster**
    - Early code generation: GPU *parallel-for* ~5% faster (GPUs), other statements and architectures a work in progress…

NNSA Support for Open Source Fortran solution for LLVM

- LLVM continuing to make strong inroads in HPC
  - High quality C/C++ (clang)
  - Basis for compiler strategy on Sierra
  - Infrastructure lacking a quality Fortran front end
- ATDM kicked off a project in FY15 to fund NVIDIA/PGI to release their production front-end Fortran compiler to LLVM community (aka “Flang”)
- Expected to be available for community feedback, input and contributions in late 2016
  - Currently being tested with a small set of alpha testers
  - Many requests to grow team from labs, industry and academia.

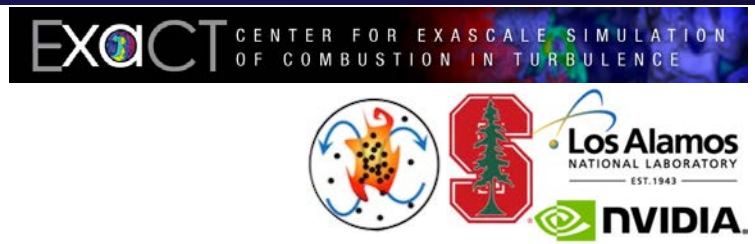
Is the key Cost Really Data Movement?

- ***"Data movement is expensive, compute is free."***
- ***But…Idle processors are not free***
  - **Trinity**: *If you dump data from memory to disk you spend 10X more power waiting on the data to move than to move the data!*
- ***No surprise… We want to keep processors busy…***

Operation	Energy (pJ)
64-bit integer operation	1
64-bit floating-point operation	20
256 bit on-die SRAM access	50
256 bit bus transfer (short)	26
256 bit bus transfer (1/2 die)	256
Off-die link (efficient)	500
256 bit bus transfer (across die)	1,000
DRAM read/write (512 bits)	16,000
HDD read/write	$O(10^6)$

Courtesy Greg Asfalk (HPE) and Bill Dally (NVIDIA)

Quick Overview of the Legion Programming Model



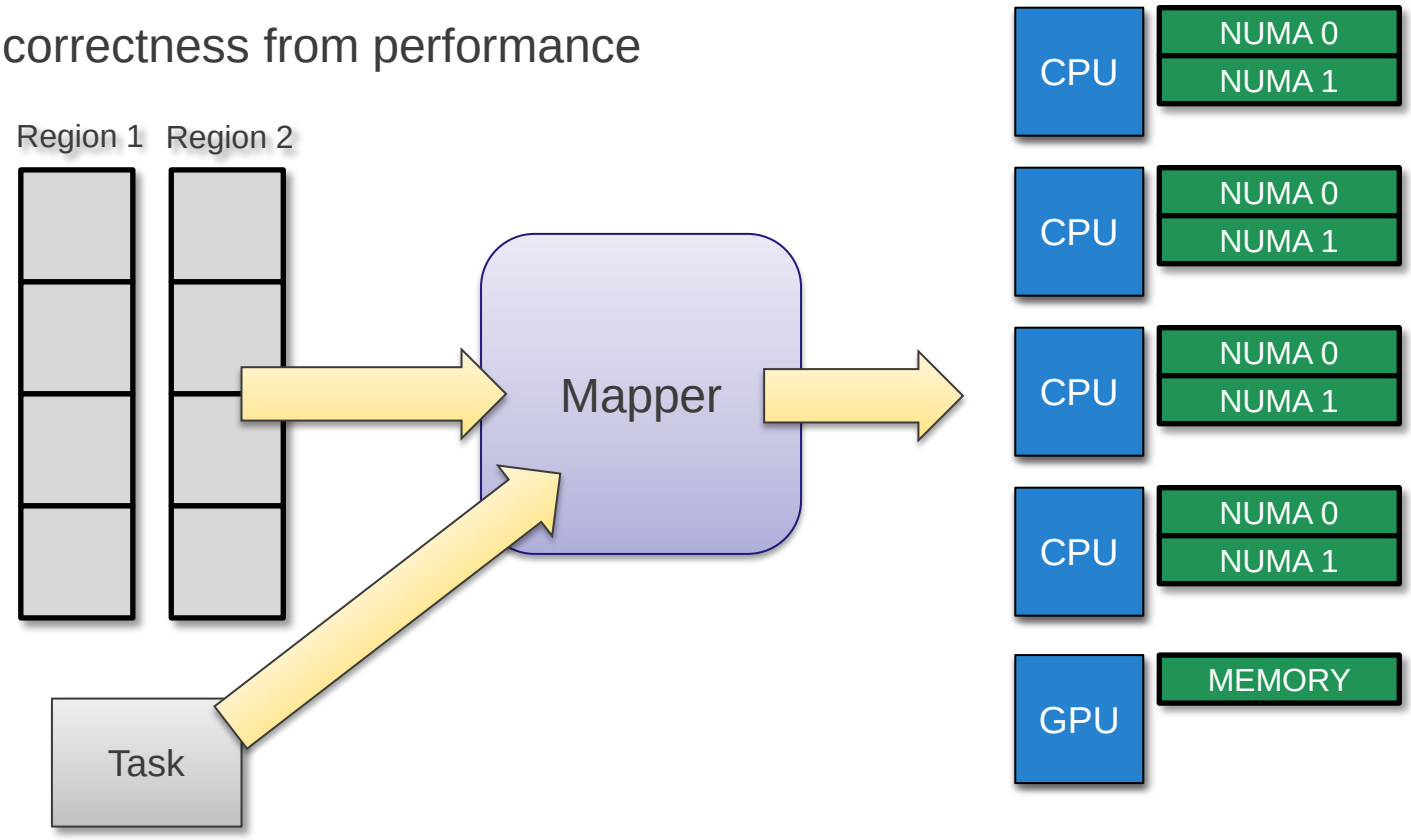
Targets heterogeneous, distributed memory machines

- **Task:** unit of parallel execution
  - Task arguments are **regions** (collection of data w/ an index and field space)
  - Regions may be arbitrarily **partitioned** (by index space) and **sliced** by field (access)
- **Tasks must specify how they use their regions:**
  - **Privlieges**(read-only,write-only,read+write,reduce)
  - **Coherence**(exclusive,atomic,simultaneous) –w/ respect to “sibling” tasks
- **Tasks launches follow sequential semantics with relaxed execution order**

https://github.com/StanfordLegion/legion

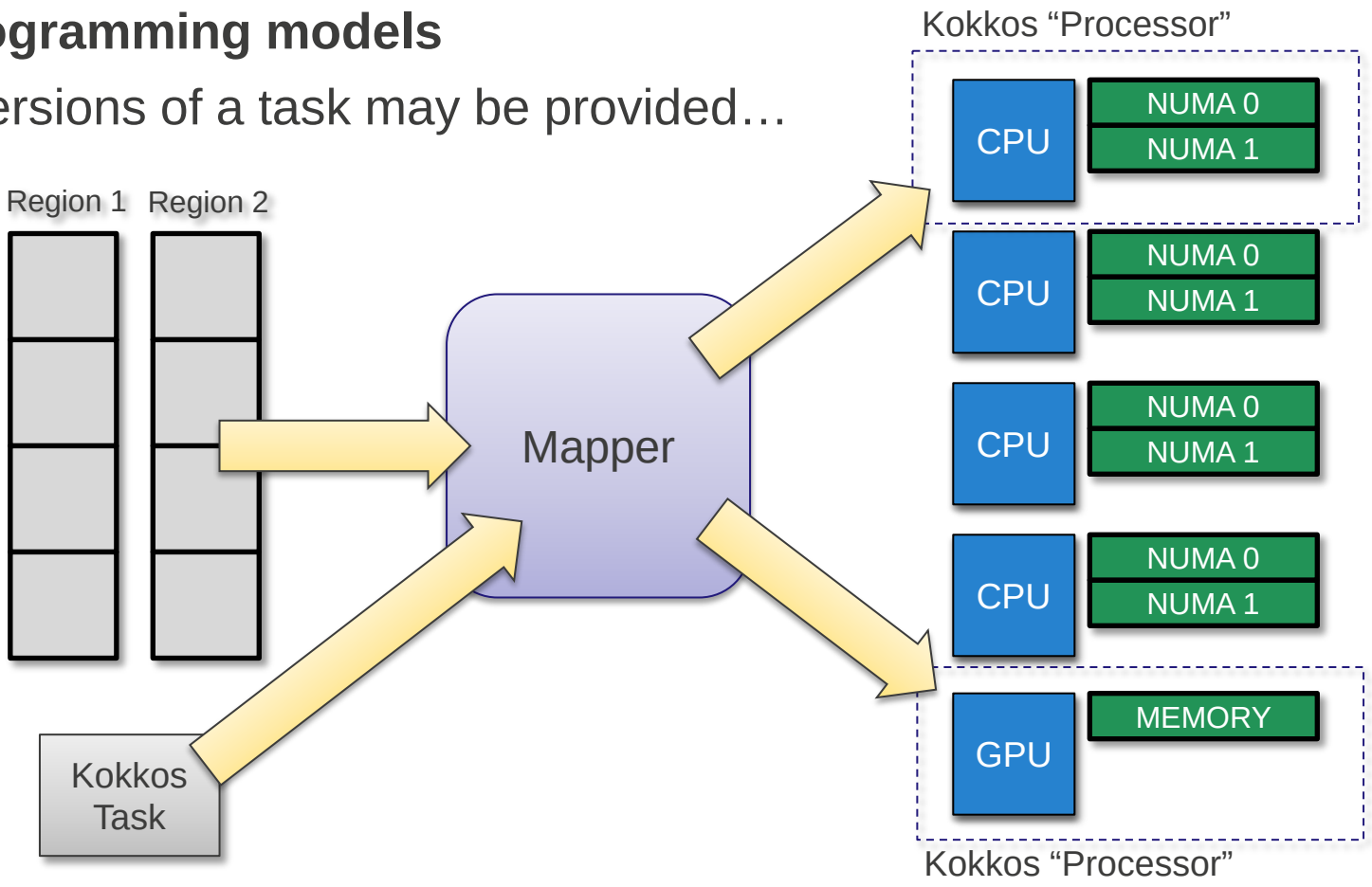
Mapping Tasks and Data to Hardware Resources

- **Application selects:**
  - Where tasks run and where regions are placed
  - Computed dynamically
  - Decouple correctness from performance



Mapping Tasks and Data to Hardware Resources
Interoperability: Supporting Task-Level Models

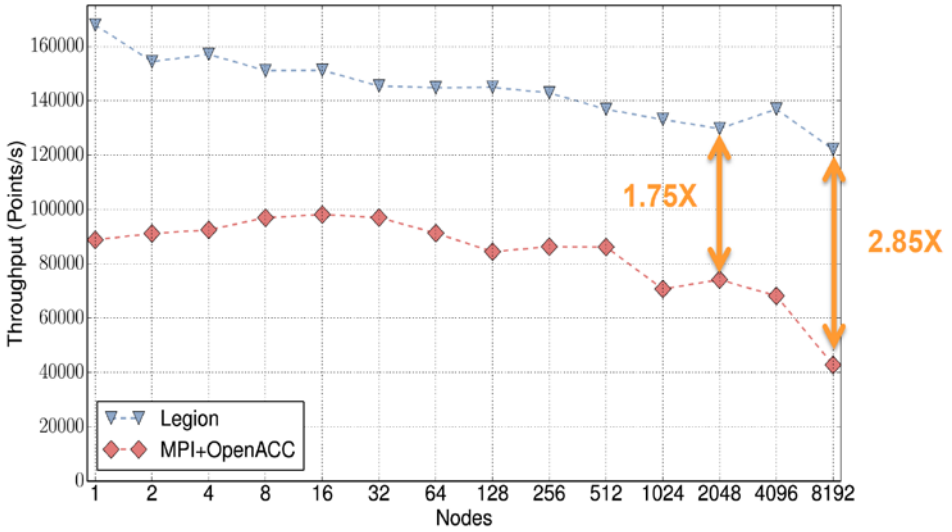
- **Interoperability: Allow tasks to be written in different programming models**
  - Different versions of a task may be provided…

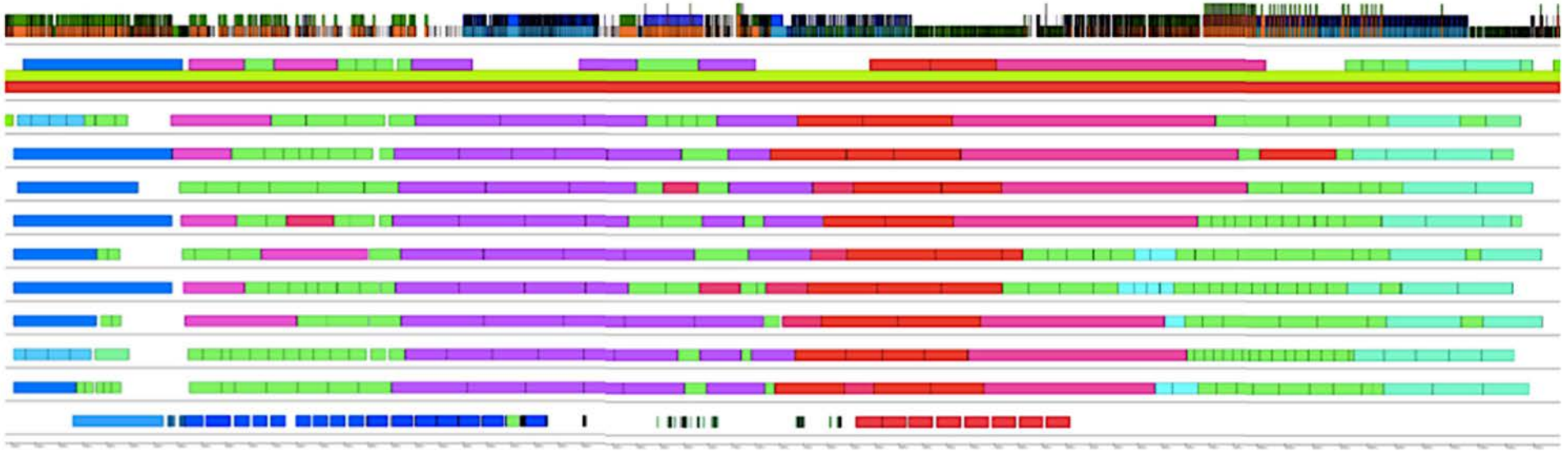


Legion S3D Execution and Performance Details

- **Mapping for 96^3 Heptane**
  - Top line shows runtime workload
  - Different species required mapping changes (e.g., due to limited GPU memory size) – i.e. tuning is often not just app and system specific…

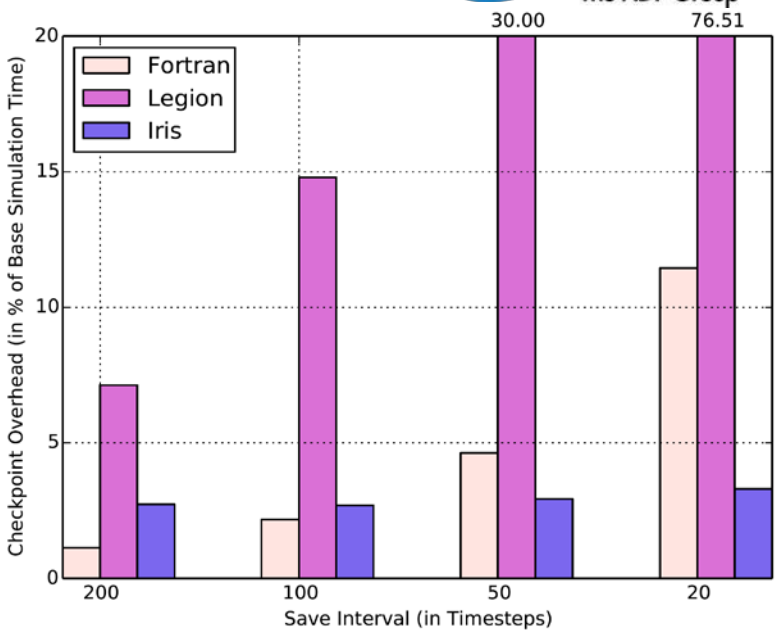
Weak scaling results on Titan out to 8K nodes





Workflow: Integration of External Resources into the Programming Model

- **We can't ignore the full workflow!**
  - Amdahl's law sneaks in if we consider I/O from tasks – 15-76% overhead vs. 2-12% of original Fortran code!
- **Introduce new semantics for operating with external resources (e.g. storage, databases, etc.).**
  - Incorporates these resources into deferred execution model
  - Maintains consistency between different copies of the same data
  - Underlying parallel I/O handled by HDF5 but scheduled by runtime
- **Allow applications to adjust the snapshot interval based on available storage and system fault concerns instead of overheads.**



Performance of S3D checkpoints running on 64 nodes (i.e., 1,024 cores) of Titan.

THANKS OLCF!

What do we do about our Legacy Codes?

- **Legacy (mostly means Fortran)**
  - Fortran actually has some nice/helpful…
- **Refactor code to use well defined, pure functions/subroutines**

```
function square(x)
  real :: x, square
  square = x * x
end function
```

```
pure function square(x)
  real, intent(in) :: x
  real :: square
  square = x * x
end function
```

- **Remaining aspects of Legion-Fortran interface is a work in progress (unfortunately details can vary between compiler implementations – one size does not fit all…)**

Sandia ASC/ATDM Level 2 milestone: Assess leading AMT runtimes to inform ATDM's technical roadmap

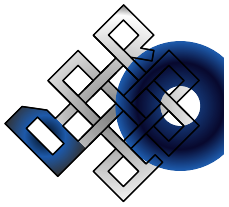

SANDIA REPORT

ASC ATDM Level 2 Milestone #5325: Asynchronous Many-Task Runtime System Analysis and Assessment for Next Generation Platforms

Sandia National Laboratories

- **Broad survey of many AMT runtime systems**
- **Deep dive on Charm++, Legion, Uintah assessing**
  - ***Programmability:*** Will this runtime enable efficient expression of our codes?
  - ***Performance:*** How performant is this runtime on current platforms and how well suited is this runtime to address future architecture challenges?
  - ***Mutability:*** What is the ease of adopting this runtime and modifying it to suit Sandia ASC/ATDM needs?

Performance Findings	Programmability + Mutability Findings
Empirical studies show an AMT runtime can mitigate performance heterogeneity inherent to the machine itself	Legion runtime needs hardening and lacks application facing API
MPI and AMT runtimes perform comparably under balanced conditions	Uintah is targeted at Cartesian structured mesh applications (ATDM requires hybrid meshing capabilities)
Previous experiments show strengths of AMT runtimes for dynamic applications	Charm++ requires new abstractions and improved component implementations to realize its full potential



DARMA is a portability layer for AMT runtimes that addresses key gaps identified in the L2 study

Sandia National Laboratories

- **DARMA serves several key purposes:**
  - Insulate applications from runtime system and machine architecture
  - Improve application programmability
  - Synthesize application requirements for HPC runtime system developers

- **DARMA is an embedded DSL (C++11/14) comprising three layers:**
  - Application-facing front end API
  - Translation layer (C++ template metaprogramming)
  - Back end API (abstract classes and functions for runtime developers to implement)

Active collaborations with industry, vendors and researchers are helping to ensure success

- **Compiler teams and programming model developers are improving support for C++ based encapsulation**
  - IBM, NVIDIA, Intel, GNU, AMD (Trinity/Sierra CoE, DesignForward, FastForward)
  - OpenMP 4.0 → OpenMP 4.5 & beyond
  - C++ Issues and feature requests coordinated between Kokkos and RAJA teams, participation in C++ ANSI Standard across labs (happy to see rest of DOE community joining us)
  - Prototyping C++ transformations in ROSE, Clang/LLVM.
- **Tool support for C++ templates and new models/tasking**
  - We have a start, need more activity…

Conclusions

Overall ATDM has used a holistic approach

- **Employing all available resources to maximize leverage**
  - Vendor research (FF/DF)
  - ATS procurements (pre-exascale architectures)
  - PSAAP2 (aggressive pursuit of exascale solutions)
  - ATDM (new codes and supporting software)
  - ASCR exascale research
  - ASC research in algorithms, software, and hardware

- **Questions?**