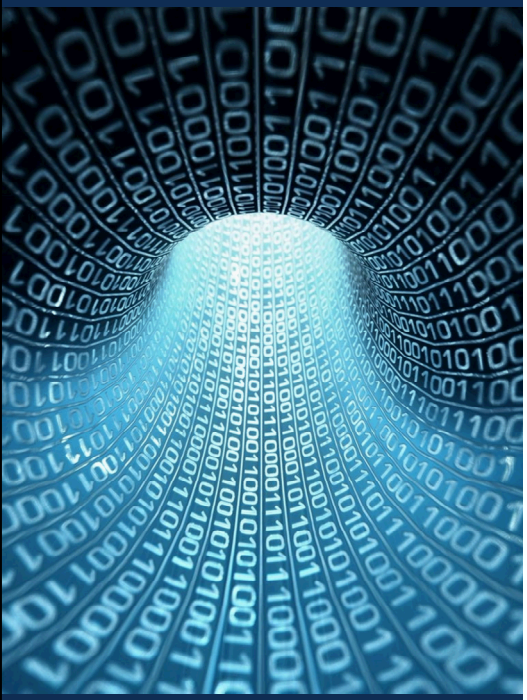


Sandia National Laboratories

Exceptional service in the national interest

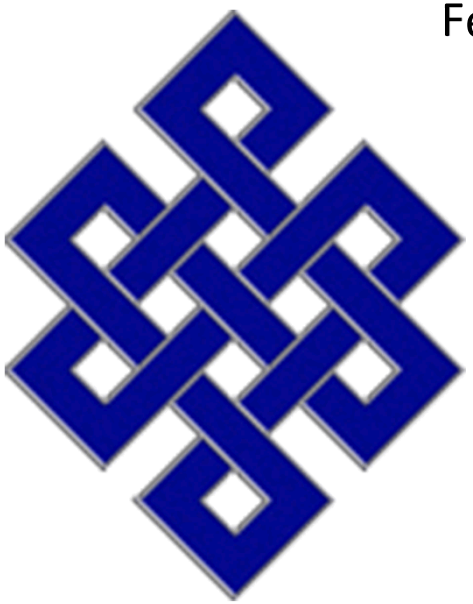
SAND2016-1322C

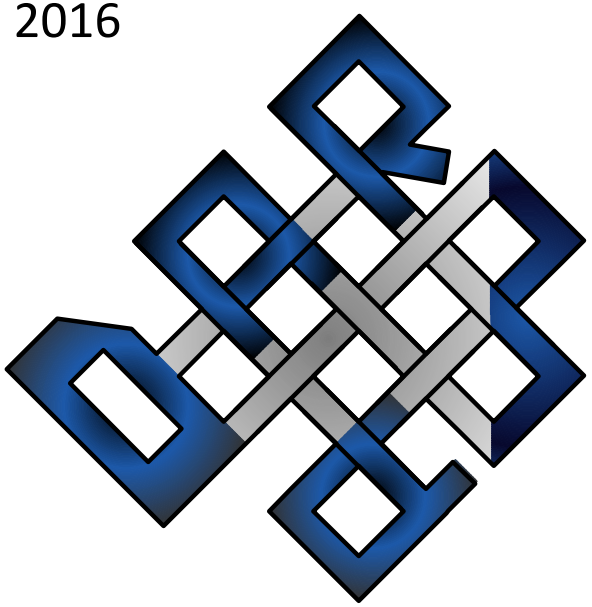
The DARMA Approach to Asynchronous Many-Task (AMT) Programming

Jeremiah J. Wilke, David S. Hollman, Nicole Slattengren, Hemanth Kolla, Francesco Rizzi, Keita Teranishi, Janine C. Bennett (PI), Robert L. Clay (PM)

Programming Models and Co-Design Meeting

Feb 3, 2016





U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

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. SAND NO. 2011-XXXXP

SAND2016-0930 C

What are core requirements for next-generation applications?

Sandia National Laboratories

- An abstraction layer between applications and the runtime system/physical architecture
- An implementation of a software stack that supports this abstraction layer
- Community best practices and eventual standards for this abstraction layer

What are core requirements for next-generation applications?

Sandia National Laboratories

- An abstraction layer between applications and the runtime system/physical architecture
- An implementation of a software stack that supports this abstraction layer
- Community best practices and eventual standards for this abstraction layer

The DARMA project at Sandia performs AMT runtime system R&D to address these core application requirements

Asynchronous Many-Task (AMT) runtimes address key performance challenges posed by future architectures

Sandia National Laboratories

- Performance challenges:
  - Utilizing whole machine requires more parallelism
  - Managing deep memory hierarchies requires flexible staging of data/assigning work
  - Handling dynamic workloads requires flexible task scheduling

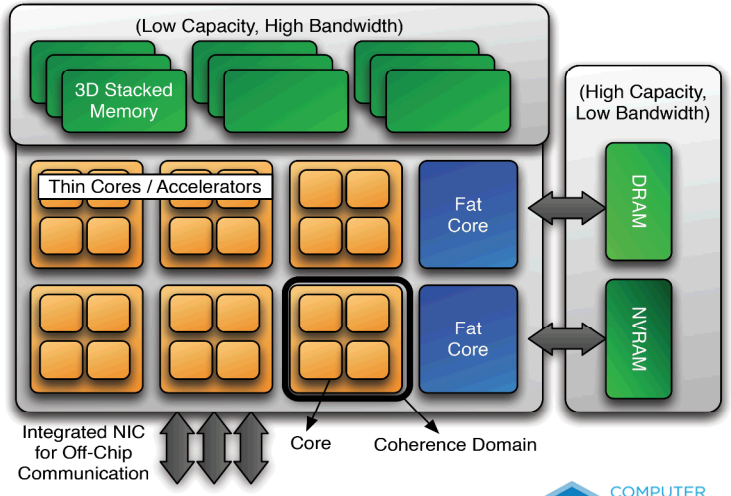


Image courtesy of www.cal-design.org

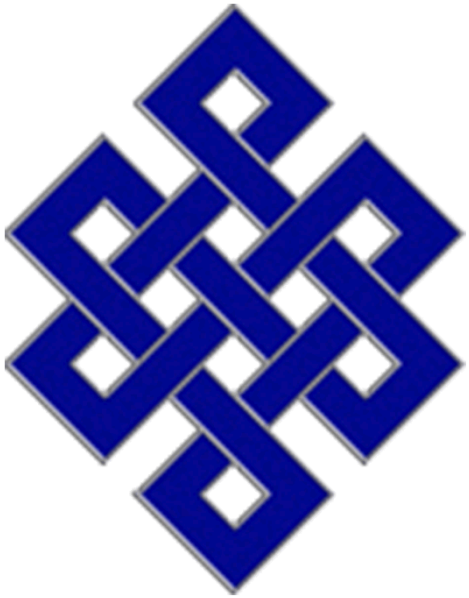
COMPUTER ARCHITECTURE LABORATORY
EXASCALE DESIGN SPACE EXPLORATION

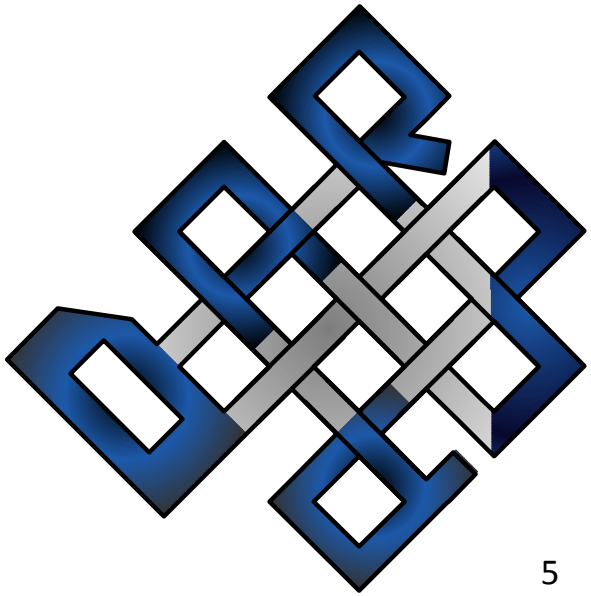
- **Asynchronous**: express all possible parallelism and minimize/hide communication/scheduling latency
- **Many-task**: Chunks of work of ``correct'' granularity that can be flexibly assigned to different memory/execution spaces

DARMA is the reincarnation of DHARMA: Origin story

Sandia National Laboratories

- ***D***istributed, ***A***synchronous, ***R***esilient ***M***odels for ***A***pplications
- Originally fault-tolerance: ***D***istributed ***H***ash ***A***rray for ***R***esilience in ***M***assively parallel ***A***pplications
- Dhr is sanskrit meaning to hold, keep
- Programming model concerns for fault-tolerance similar to AMT
  - Simplify reasoning about code correctness
  - Latency hiding
  - Recoverable (migratable) chunks of work

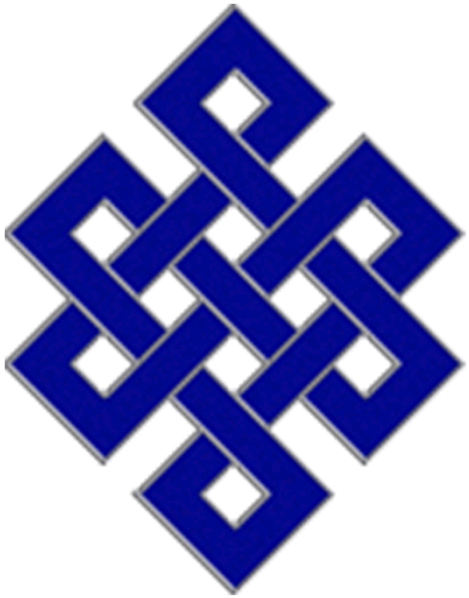


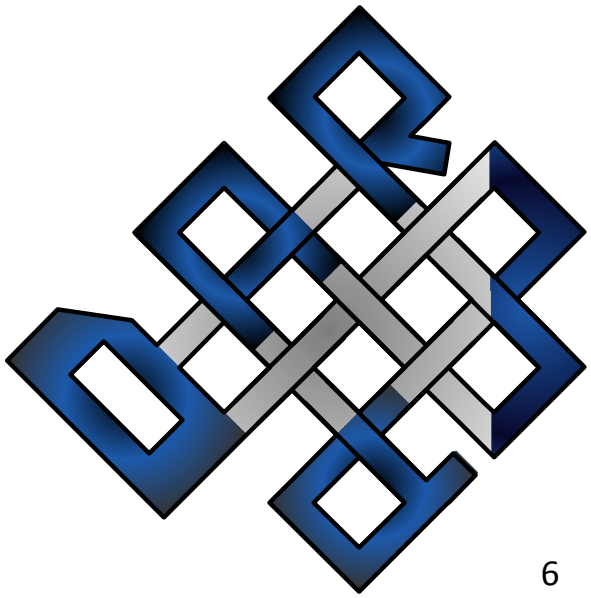


DARMA is the reincarnation of DHARMA: Origin story

Sandia National Laboratories

- ***D***istributed, ***A***synchronous, ***R***esilient ***M***odels for ***A***pplications
- Originally fault-tolerance: ***D***istributed ***H***ash ***A***rray for ***R***esilience in ***M***assively parallel ***A***pplications
- Dhr is sanskrit meaning to hold, keep
- Programming model concerns for fault-tolerance similar to AMT
  - Simplify reasoning about code correctness
  - Latency hiding
  - Recoverable (migratable) chunks of work





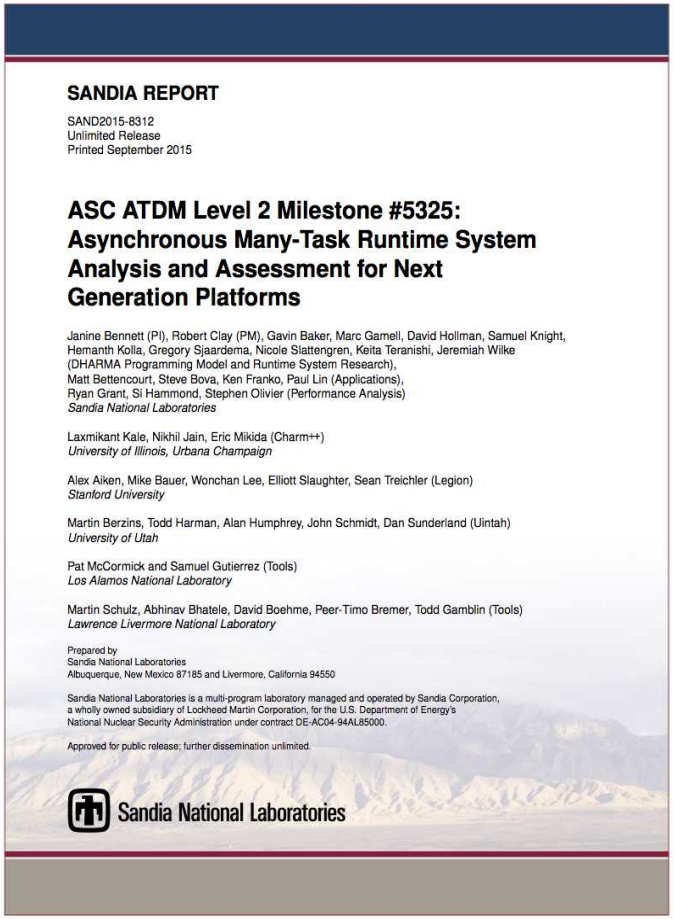
Noble Truths of HPC

- All life is suffering
- Our desire for more flops is the source of our suffering

Sandia led a comparative analysis study of leading AMT runtimes to inform our technical roadmap

Sandia National Laboratories

- Broad survey of many AMT runtime systems
- Deep dive on Charm++, Legion, Uintah
- Programmability
  - Does this runtime enable efficient expression of our workloads?
- Performance
  - How performant is this runtime for our workloads on current platforms?
  - How well suited is this runtime to address exascale challenges?
- Mutability
  - What is the ease of adopting this runtime and modifying it to suit our needs?



SANDIA REPORT

SAND2015-8312
Unlimited Release
Printed September 2015

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

Janine Bennett (PI), Robert Clay (PM), Gavin Baker, Marc Gamell, David Hollman, Samuel Knight, Hemanth Kolla, Gregory Sjaardema, Nicole Slattengren, Keita Teranishi, Jeremiah Wilke (DHARMA Programming Model and Runtime System Research), Matt Bettencourt, Steve Bova, Ken Franko, Paul Lin (Applications), Ryan Grant, Si Hammond, Stephen Olivier (Performance Analysis)
Sandia National Laboratories

Laxmikant Kale, Nikhil Jain, Eric Mikida (Charm++)
University of Illinois, Urbana Champaign

Alex Aiken, Mike Bauer, Wonchan Lee, Elliott Slaughter, Sean Treichler (Legion)
Stanford University

Martin Berzins, Todd Harman, Alan Humphrey, John Schmidt, Dan Sunderland (Uintah)
University of Utah

Pat McCormick and Samuel Gutierrez (Tools)
Los Alamos National Laboratory

Martin Schulz, Abhinav Bhatele, David Boehme, Peer-Timo Bremer, Todd Gamblin (Tools)
Lawrence Livermore National Laboratory

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

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.

Approved for public release; further dissemination unlimited.

Sandia National Laboratories

Vendor-supported runtime system and standards are ideal but AMTs are still an active research area

Sandia National Laboratories

We face a spectrum of choices/risks in developing technical roadmap

Build system from scratch and take ownership

Risk: potential lack of vendor support/buy in

Lots of control, but lots of extra investment

Rely completely on external partners

Risk: current academic runtimes may lack features to support our workloads

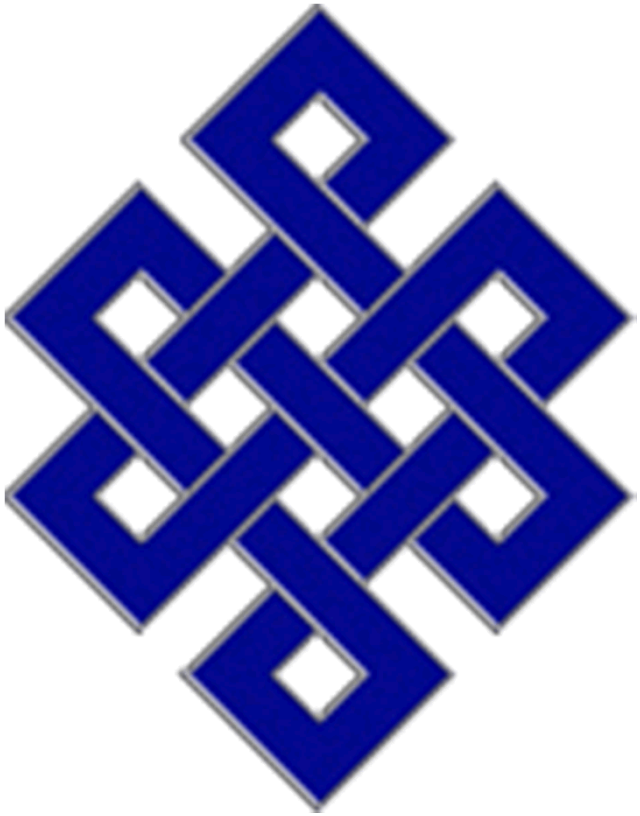
Less control, but less investment

Lessons learned from study led to application-driven programming model specification co-design effort

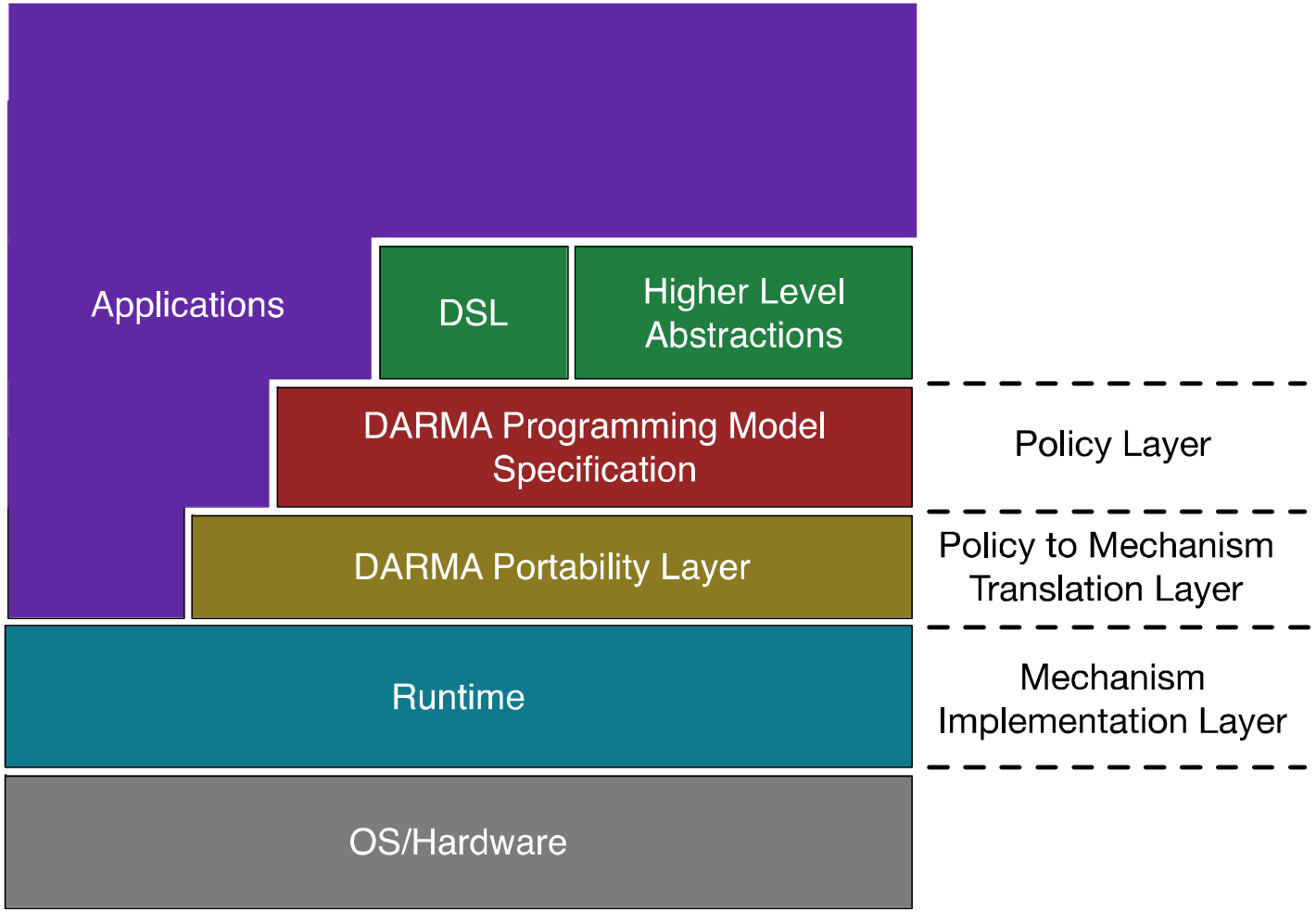
Sandia National Laboratories

- Data, task, and pipeline parallelism can be expressed in different ways
  - Explicit parallelism vs apparently sequential semantics
  - Arbitrary data structures vs strong data model
  - Runtime vs user-level control
  - New language vs embedded in C/C++
- Model should enhance performance, productivity, resilience
  - Applications should not be (much) more difficult to write than MPI
  - Make difficult things more tractable, e.g. load balancing, fault-tolerance
- Design space tradeoffs need further assessment prior to committing to a single runtime
  - Across variety of applications and architectures
  - Further research required in some aspects of runtime (e.g., resource management)

The exascale software stack should ideally NOT look like this



DARMA software stack separates policy and mechanism



Policy: Express correctness and performance requirements

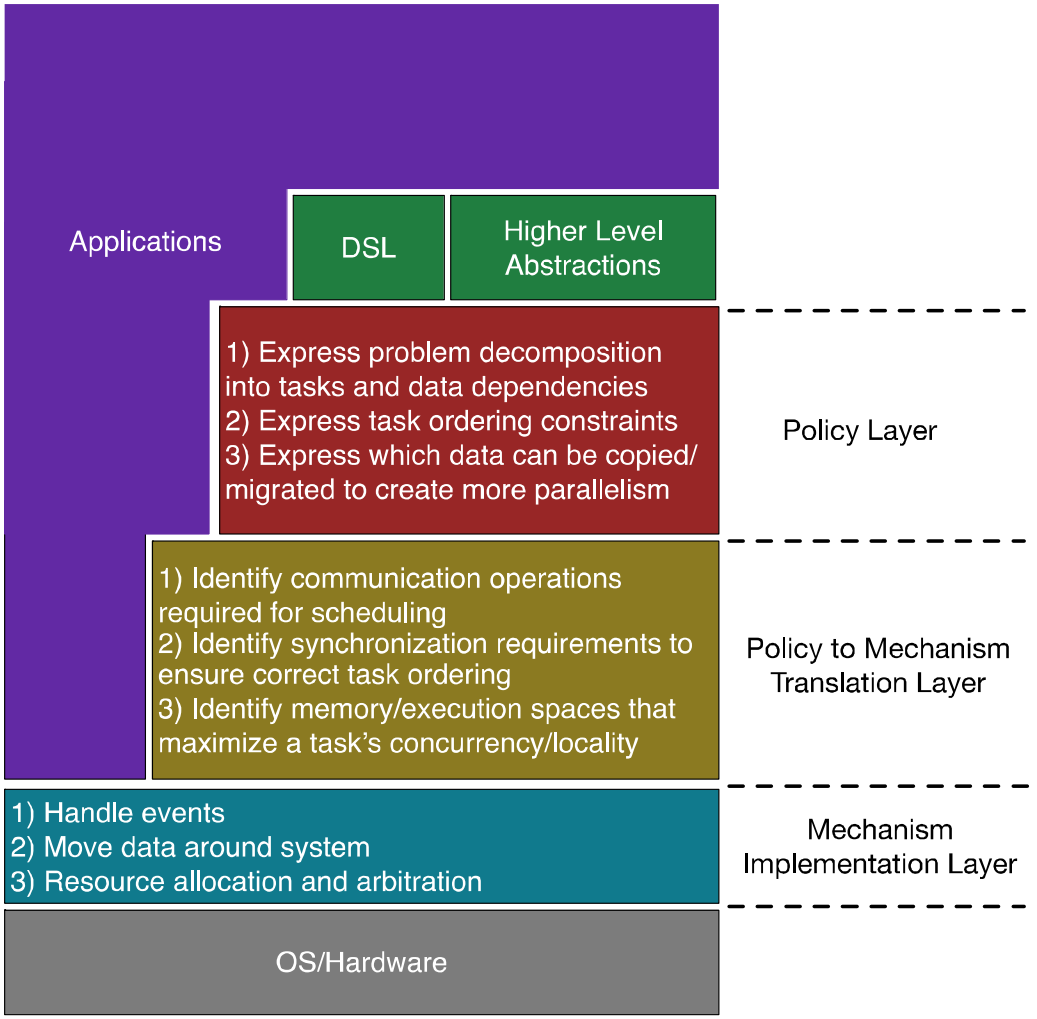
Mechanism: Implement correctness and performance requirements

Expression of policy enables runtime freedom to make complex performance-oriented decisions

Sandia National Laboratories

Design Intent:

- Applications specify policy
  - Enable rapid development of correct implementation
- Applications can specify mechanism
  - Enable improvement towards performant implementation



What do we mean by "application-driven co-design of a programming model specification"?

Sandia National Laboratories

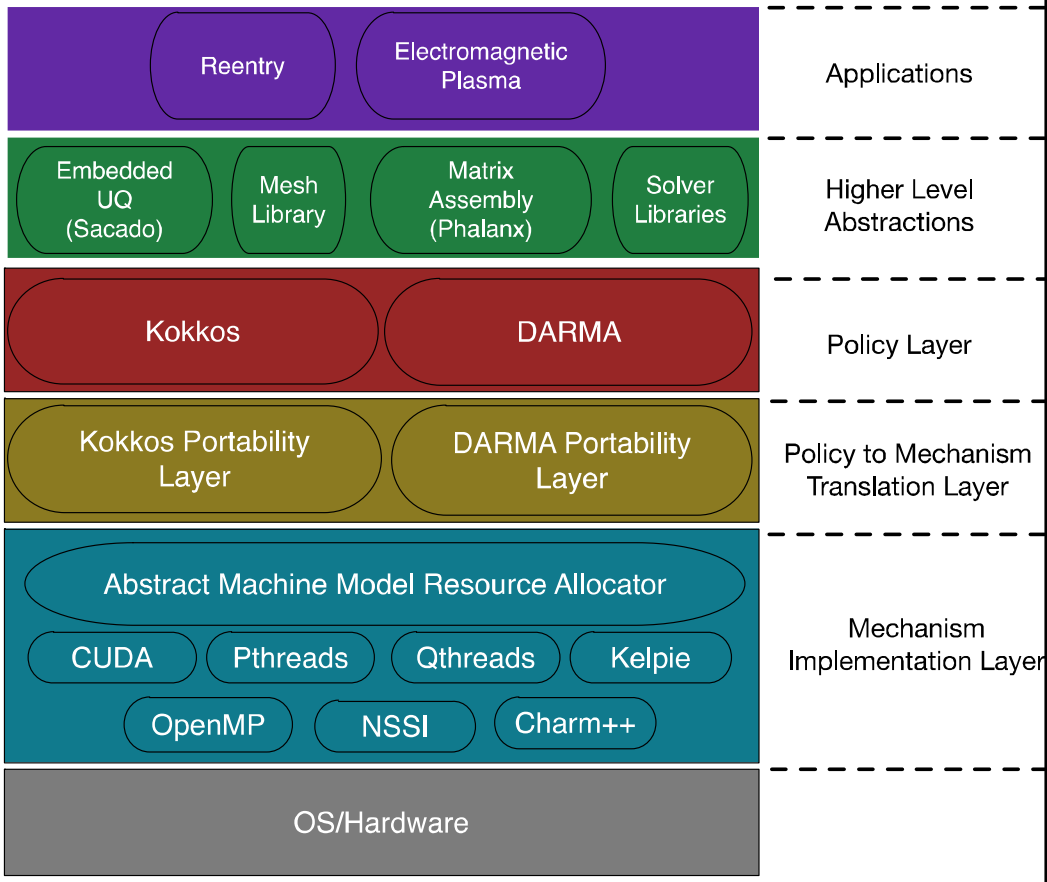
- Runtimes can be decomposed into a specification and implementation
  - The specification is formal documentation that
    - Provides abstractions for expressing what an application does (i.e., the programming model)
    - Can express correctness requirements
    - Can express performance requirements
  - The implementation maps specification requirements onto specific operations/events
- We are co-designing an AMT programming model specification with application and runtime developers
  - Meet our application requirements
  - Generate the runtime requirements

The separation of policy and mechanism facilitates exploration of runtime design space

Sandia National Laboratories

- AMT software stack working group at Sandia
  - DARMA
  - Kokkos
  - Data Warehouse/Kelpie
  - Resource allocation and management
  - Qthreads
- Initial implementation of stack this year leveraging Charm++
- Working with community to explore alternative stack implementations
  - OCR, REALM

Sample Sandia Software Stack



The Ontology of DARMA: Axioms/assumptions of programming model derived from L2/co-design study

Sandia National Laboratories

- SPMD is the dominant parallelism
- There will too much compute (parallelism) available in the hardware for basic data parallelism to fill
- Extra asynchrony should not complicate reasoning about application correctness (intuitive semantics, debugging tools)
- The traditional MPI abstract machine model (uniform compute elements, flat memory spaces) will get further and further away from actual system architecture
- There exist many applications/algorithms with dynamic load balance, dynamic sparsity, or complex workflow coupling whose development would be greatly accelerated by a more productive programming model

Keep simple things simple, keep tractable things tractable, make difficult things tractable

Sandia National Laboratories

- Simple/tractable
  - SPMD launch and initial problem decomposition/distribution
  - Collectives
  - Basic checkpoint/restart fault recovery supported
  - Application-specific data structures/layouts
- Difficult
  - Express/mix all forms of parallelism (data, pipeline, task)
  - Dynamic load balancing, work stealing
  - Data staging (software-managed cache)
  - Performance portability across execution spaces
  - Macro data-flow parallelism (parallelism within a task)

The DARMA programming model specification is a set of parallel semantics embedded in C++ syntax

Sandia National Laboratories

- Coordination semantics replace explicit send/recv
- Enqueue work to be performed instead of explicitly (imperatively) doing work immediately or blocking
- As much as possible, preserve sequential semantics to simplify reasoning about code correctness
- *Use standard C++ constructs* (e.g., reference-counted pointers) to manage parallelism
- Do not need to know C++11 to code
  - Works with gcc >= 4.7, clang >= 3.5

Extended coordination semantics enables explicit SPMD parallelism

Sandia National Laboratories

```
MPI_Send(...) -> publish(key={...}, readers={...})
MPI_Recv(...) ->  read(write)_access(key={...})
```

- Global memory space identifying migratable data with explict tuple identifiers
- Extra concept of “readership” and “versions” allows efficient data reuse, zero-copy transfers, in-place updates
- “Explicitly parallel” semantics encourage correct parallel execution, but do not guarantee it
- Tuples are the only required “data model”
- Express dependencies on coarse-grained chunks, but allow fine-grain slicing on reads

Apparently sequential semantics with permission qualifiers/task blocks enables task/pipeline parallelism

Sandia National Laboratories

- Use read/write qualifiers to automatically generate task parallelism that enforces sequential semantics
  - Legion, OmpSs, PARSEC, OpenMP 4.0, etc do similar things
- Implicit, local tuple space identifiers based on task order
- Embedded in C++11 syntax (no pragmas)
- Deferred execution through external functors or in-line lambdas – removes need for Isend/Wait

```
int run(){
  A a = read_access(...);
  B b = write_access(...);
  task<fxn_name>(a,b); (1),(2)
  inline_task { (3),(4),(5)
    b += a
  }
```

① Create task from fucntor `fxn_name`
② Log permissions required by task
③ Convert code block to schedulable task (similar to Cilk, but uses C++11 lambdas)
④ Automatically determine read/write permissions through C++11
⑤ Automatically analyze read/write conflicts with other tasks

Quick summary of programming model design

Sandia National Laboratories

- Application controls initial problem decomposition/distribution through coordination
  - Explicit parallelism at user-level
- Extra task/pipeline parallelism added through read/write qualifiers and task annotations
  - Implicit parallelism compliant with sequential semantics
  - More runtime responsibility. Easier to reason about code correctness.
- Embedded in C++11
- But no template metaprogramming apparent to user
- Complementary to Kokkos/Raja – specifies different policies

The DARMA project has three closely-coupled key activities

Sandia National Laboratories

- Co-design AMT programming model specification
  - Gather application requirements for programming model/runtime
  - Assess what runtime requires programming model/application to express
  - Limit scope of what pieces of software stack WE need to implement
- Implement specification
  - Leverage existing efforts
  - Encourage vendor involvement
- Work with community to define best practices and eventual standards for AMT
  - Collaborating with Tim Mattson's team at Intel on DARMA programming model specification
  - Recurring engagement with Charm++, OCR, Legion teams

Let us know if you are interested in collaborating in any of these areas!