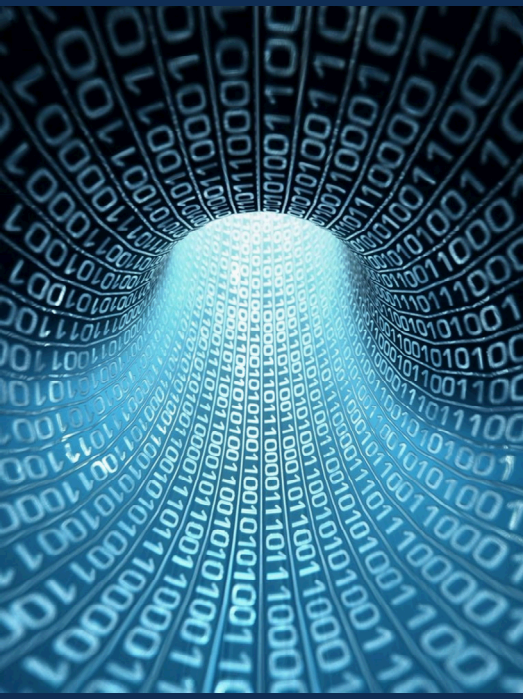
SAND2015-5106PE



Asynchronous Many-Task Programming Models for Next Generation Platforms

Jeremiah Wilke, Ken Franko, David Hollman, Samuel Knight, Hemanth Kolla, Paul Lin, Greg Sjaardema, Nicole Slattengren, Keita Teranishi

Janine Bennett (PI), Robert Clay (PM)

Sandia National Laboratories

Exceptional service in the national interest

U.S. DEPARTMENT OF ENERGY

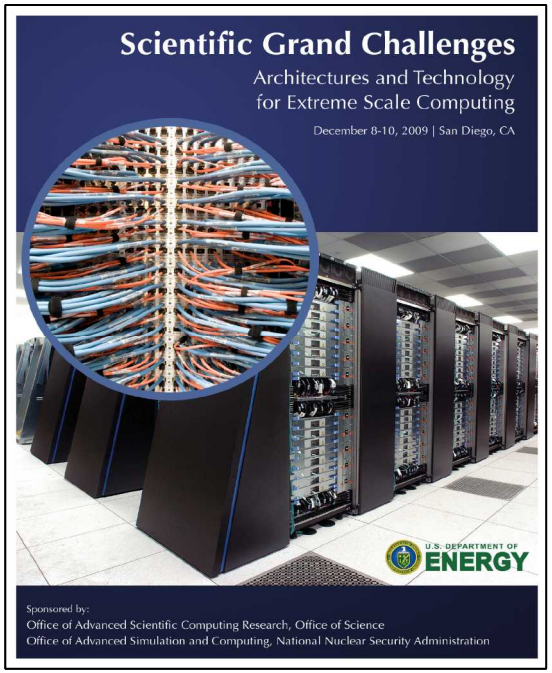
NNSA National Nuclear Security Administration

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

We expect a lot from our programming models

Sandia National Laboratories

- Expressive
  - Data parallelism: Same computation on different data
  - Task parallelism: Different computations on same or different data
- Performant
- Performance portable
  - Abstractions separate code specification from optimization for different architectures
- Future-proof
  - How much of the application code needs to be rewritten when moving to next generation architectures?



Exascale introduces significant complexities

Sandia National Laboratories

Challenges

- Increases in concurrency
- Deep memory hierarchies
- Increased fail-stop errors
- Performance heterogeneity
  - Accelerators
  - Thermal throttling
  - General system noise
  - Responses to transient failures

Overarching abstract machine model of an exascale node

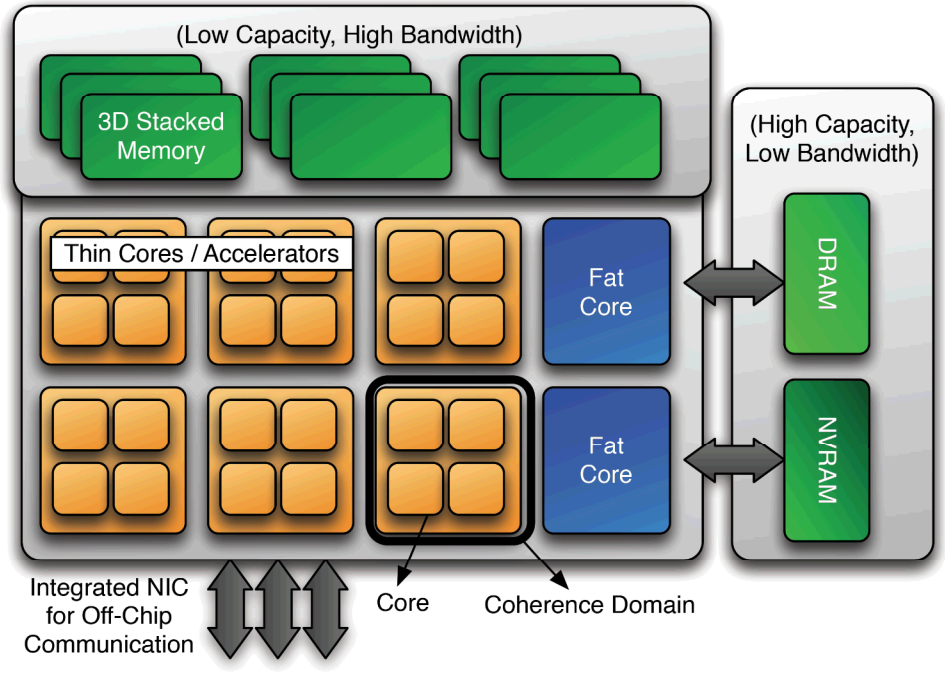


Image courtesy of www.cal-design.org

COMPUTER ARCHITECTURE LABORATORY
EXASCALE DESIGN SPACE EXPLORATION

Bulk synchronous MPI+X does not address all the challenges posed by the exascale machine model

Sandia National Laboratories

Challenges
Increases in concurrency
Deep memory hierarchies
Increased fail-stop errors
Performance heterogeneity ▪ Accelerators ▪ Thermal throttling ▪ General system noise ▪ Responses to transient failures

- Complexity of application code increases with proposed solutions

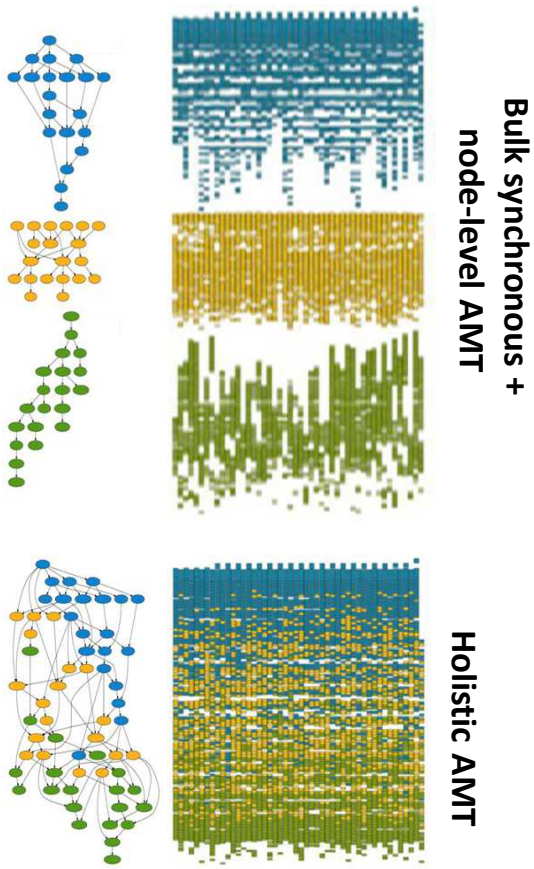
- Over-decomposition on node can help but does not solve the problem
- Algorithmic research required

Asynchronous many-task (AMT) programming models show promise against exascale challenges

Sandia National Laboratories

- Runtime systems show promise at sustaining performance despite node-degradation and failure
- Directed Acyclic Graph (DAG) of Tasks
- Active area of research
  - Charm++, DAGuE, DHARMA, HPX, Legion, OCR, STAPL, Uintah, …

Images courtesy of Jack Dongarra

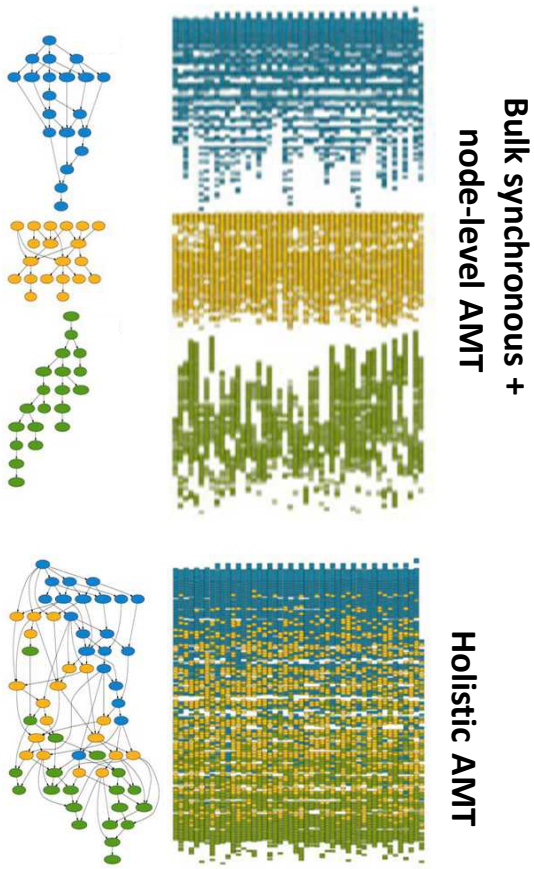


Asynchronous many-task (AMT) programming models show promise against exascale challenges

Sandia National Laboratories

- Runtime systems show promise at sustaining performance despite node-degradation and failure
- Directed Acyclic Graph (DAG) of Tasks
- Active area of research
  - Charm++, DAGuE, DHARMA, HPX, Legion, OCR, STAPL, Uintah, …

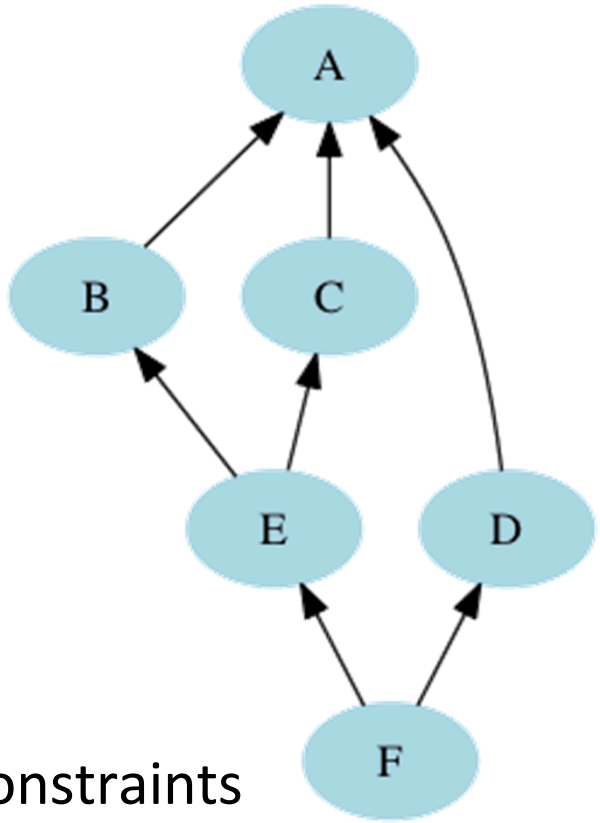
Images courtesy of Jack Dongarra



? Expressive ? Performant ? Portable ? Future-Proof

AMT programming models facilitate the expression of task and data parallelism

Sandia National Laboratories



Every program has a task graph, implicitly or explicitly!

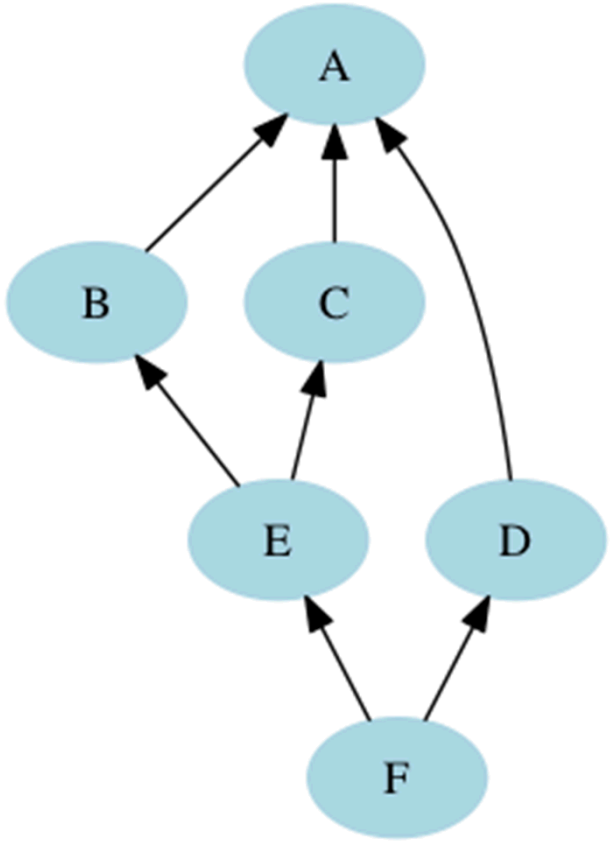
Task graph
Nodes: Tasks
Edges: Precedence constraints

✓ **Expressive** ? Performant ? Portable ? Future-Proof

Imperative coding style can be easier conceptually, but defines strict execution

Sandia National Laboratories

```
Output sequential()
{
  Output A = do_A();
  Output B = do_B(A);
  Output C = do_C(A);
  Output D = do_D(A);
  Output E = do_E(B,C);
  Output F = do_F(D,E);
  return F;
}
```

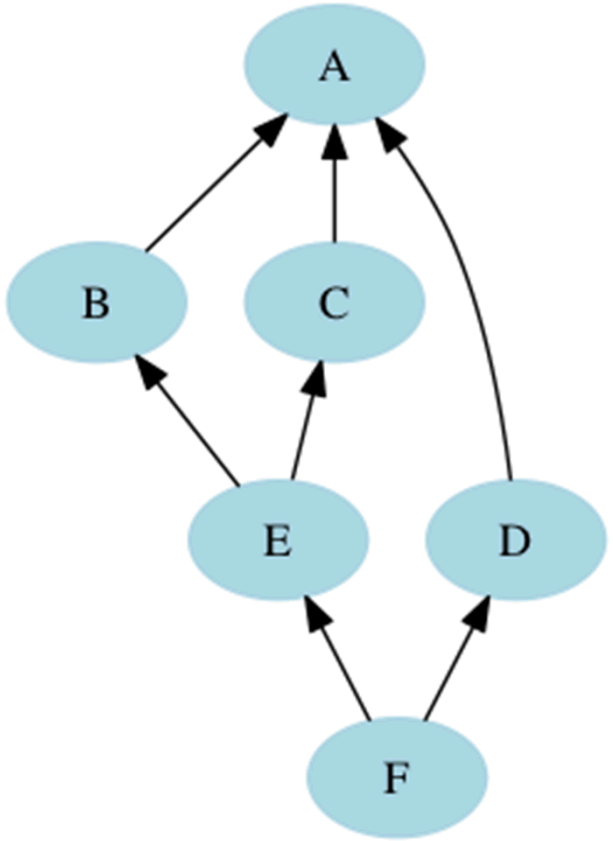


✓ **Expressive** ? Performant ? Portable ? Future-Proof

Declarative coding style describes program, flexible execution by task runtime

Sandia National Laboratories

```
Output declarative()
{
  TaskDag dag;
  Task A;
  dag.add(A);
  Task B(A), C(A), D(A);
  dag.add(B,C,D);
  Task E(B,C);
  dag.add(E);
  Task F(E);
  dag.add(F);
  dag.run();
  return dag.result();
}
```



✓ **Expressive** ? Performant ? Portable ? Future-Proof

Declarative coding style describes program, flexible execution by task runtime

Sandia National Laboratories

```
Output declarative()
{
  TaskDag dag;
  Task A;
  dag.add(A);
  Task B(A), C(A), D(A
  dag.add(B,C,D);
  Task E(B,C);
  dag.add(E);
  Task F(E);
  dag.add(F);
  dag.run();
  return dag.result();
}
```

- Execution is deferred
- DAG runtime can
  - Detect parallelism
  - Maximize concurrency
  - Pipeline communication
  - Load balance/work-steal
  - Move work to data
  - Do transparent fault-tolerance

✓ **Expressive** ✓ **Performant** ✓ **Portable** ? Future-Proof

Distinction between programming and execution models is important in the context of portability

Sandia National Laboratories

- **Programming model**: API and abstractions in **application**
  - Do we code via procedure calls with pointers and scalar types? Or do we code via task declarations on logical handles?
  - Can be performance portable
- **Execution model**: API and abstractions in **runtime system (RTS)**
  - Does the code run in a sequential, step-by-step way? Event-driven?
  - Performance portable only within very similar architectures

✓ **Expressive** ✓ **Performant** ✓ **Portable** ? Future-Proof

AMT programing models shield application code from dynamic imablance/faults

Sandia National Laboratories

- Task graph encodes provenance information for applications
- Logical identification of tasks and data facilitates data and task migration, enabling transparent load-balancing and fault-recovery via runtime when nodes degrade or fail

```
int mpi_main()
{
  int rc;
  rc = MPI_Irecv(A0);
  if (rc == PARTNER_FAILED){
    //do something - but what?
  }
  rc = MPI_Irecv(B0);
  if (rc == PARTNER_FAILED){
    //do something - but what?
  }
  rc = MPI_Wait(A0_request);
  if (rc == PARTNER_FAILED){
    //do something - but what?
  }
```

If statements handling dynamic faults/performance variation can complicate MPI applications

✓ **Expressive** ✓ **Performant** ✓ **Portable** ✓ **Future-Proof**

DHARMA project: Distributed asyncHronous Adaptive and Resilient Models for Applications

Sandia National Laboratories

- **Project Mission:** Assess & address fundamental challenges imposed by the need for performant, portable, scalable, fault-tolerant programming models at extreme-scale
  - Assess rich feature sets/usability/performance of existing AMT runtimes in the context of Sandia workloads
  - Research in programmability, dynamic load-balancing, and fault-tolerance of AMT runtimes

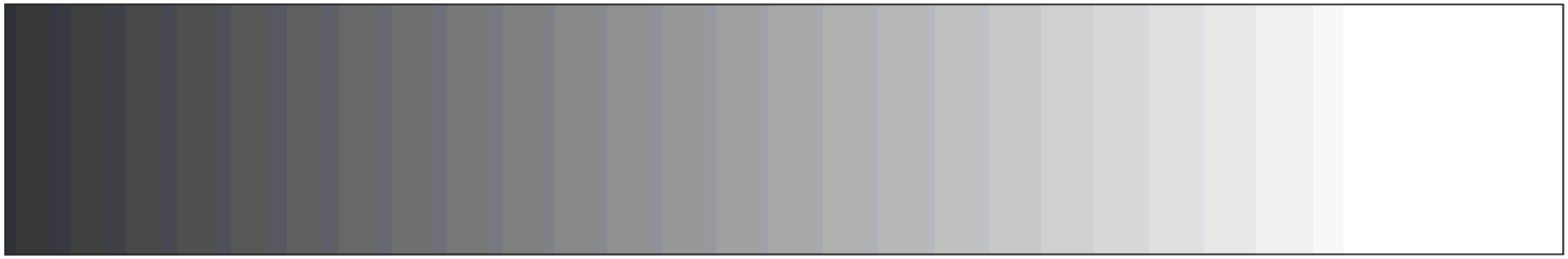
DHARMA is a fundamental Hindu concept referring to
- the order and custom which make life and a universe possible
- the behaviors appropriate to the maintenance of that order

The classical Sanskrit noun DHARMA derives from dhr
- meaning to hold, maintain, keep

Sandia faces spectrum of choices/risks in developing technical roadmap

Sandia builds system from scratch and takes ownership

Sandia relies completely on external academic partners



Lots of control, but lots of extra investment

Less control, but less investment

Comparative analysis of leading AMT RTS

Sandia National Laboratories

- We considered many runtimes over the summer of FY14
  - Charm++, Legion, Uintah, STAPL, HPX, OCR, Swift/T
- We settled on Charm++, Legion, Uintah as our top three study candidates
  - Demonstrated science applications at scale
  - Maturity of runtime
  - Three very different implementations, APIs, sets of abstractions
  - Accessibility of teams
- Coding Boot Camps with each University to develop applications in conjunction with runtime experts
- Both Sandia runtime and application developers are involved in the study

SCI
www.sci.utah.edu

Los Alamos
NATIONAL LABORATORY
EST. 1943

NVIDIA

PPL UIUC
PARALLEL PROGRAMMING LABORATORY

Comparative analysis of leading AMT RTS

Sandia National Laboratories

- Mini Aero: 3-D, unstructured, finite-volume, computational fluid dynamics code
  - Runge-Kutta fourth-order time marching
  - Options for first or second order spatial discretization
  - Includes inviscid Roe and Newtonian viscous fluxes
  - Baseline application is approximately 3800 lines of C++ code using MPI+Kokkos
- Three primary evaluation criteria
  - **Programmability:** largely qualitative feedback from application developers
  - **Mutability:** characterization of software stack, partnership strategies,...
  - **Performance studies:** strong and weak scaling, load-balancing under system heterogeneity, task- and data-granularity, power/performance

On-going experiments (summer of FY15)

Sandia National Laboratories

- Strong and weak scaling studies with varying levels of over-decomposition
- Load-balancing in the presence of system performance heterogeneity
  - Introduce performance heterogeneity via fine-grained controls on Shepard (Sandia Advanced Technology Test bed)
  - Correlate power measurements to on-node performance counters
- Assessing the role of task-granularity and its implications
  - In-situ (same resources, no copies) vs. In-transit (forced copies to shared resources) data analysis
  - Study the effects of task granularity
- Exploring use of performance analysis tools
  - Insights into performance of runtimes
  - Analysis of state of tools themselves

Initial qualitative assessment shows key features missing from leading runtimes

Sandia National Laboratories

- Uintah
  - \- Limited to structured meshes
  - \+ Framework easy to code to, familiar
- Charm++
  - \- Task-graph not dynamic, specified at compile time
  - \- Templated programming not well supported
  - \+ Load-balancing and fault tolerance are very easy!
- Legion
  - \- Requires use of their data structures
  - \- Overhead of data-driven runtime is a detractor for our dynamic workloads (Particle in Cell, Contact applications)
  - \+ Data-driven runtime very powerful for other applications, dynamic task-graphs, good architectural and conceptual design

Initial conclusions from study

Sandia National Laboratories

The cons:

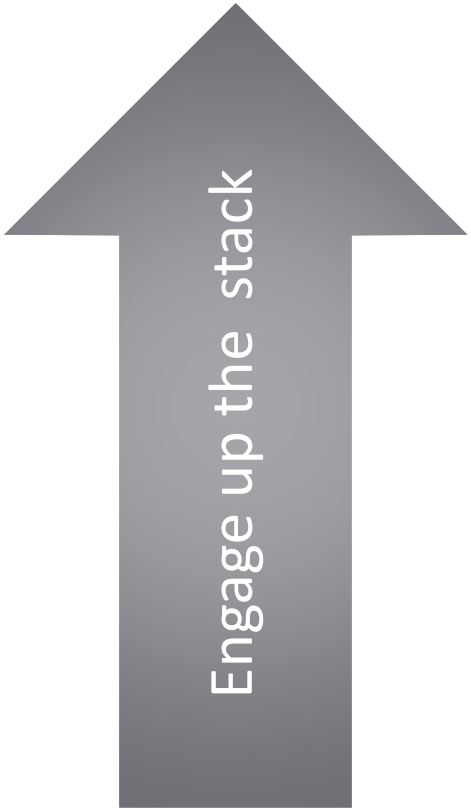
- All leading AMT programming models and RTS have been designed or demonstrated on a limited set of applications
- None of the runtimes are production ready
- None of the runtimes appear to satisfy all requirements of our application workloads (definition of requirements is still in progress)

The pros:

+ AMT runtimes show tremendous potential for addressing exascale challenges
+ The collective research being performed by AMT RTS developers is a critically important precursor to establishing community standards

Where does the AMT programming model and RTS community go from here?

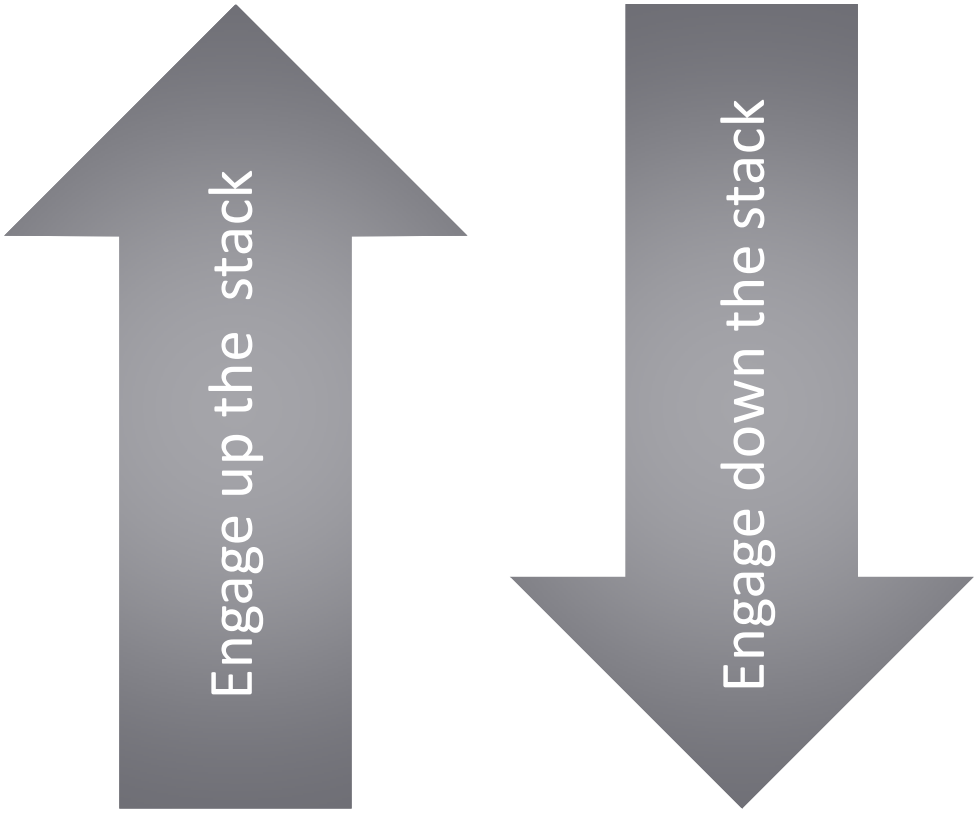
Sandia National Laboratories



Gather requirements from a broad set of HPC applications

Where does the AMT programming model and RTS community go from here?

Sandia National Laboratories

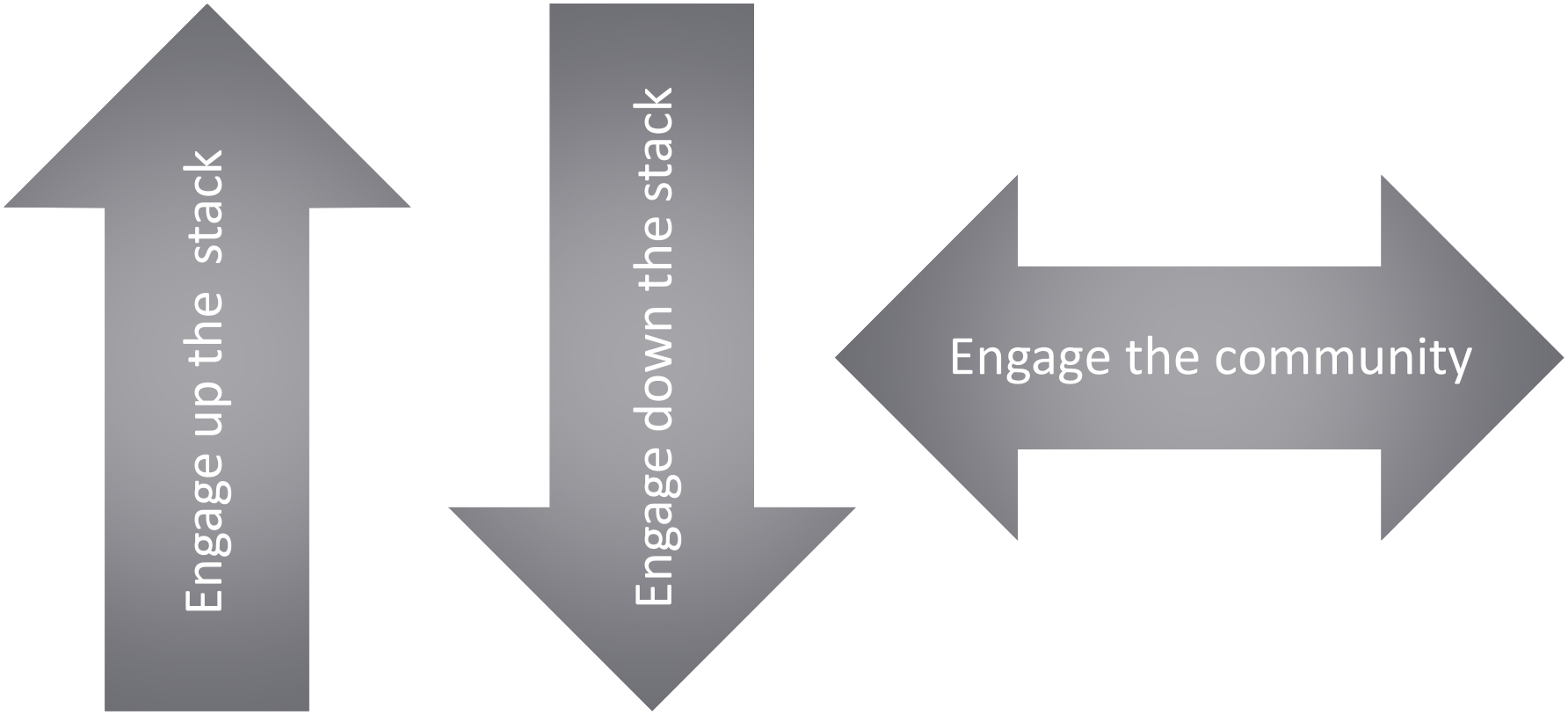


Gather requirements from a broad set of HPC applications

Communicate requirements to OS software developers

Where does the AMT programming model and RTS community go from here?

Sandia National Laboratories



Gather requirements from a broad set of HPC applications

Communicate requirements to OS software developers

Establish common terminology and best practices with an eye towards eventual standards

Sandia's ongoing and future efforts

Establish common terminology and best practices with an eye towards eventual standards

- Finish & share comparative analysis study
- Detailed report establishing terminology
  - Based on survey of 15+ parallel frameworks
  - Community feedback will be sought before finalizing
- DHARMA programming model research
  - Programming model specification
    - ***expressive*** enough to support many execution models (using runtime/compile time switches)
  - Build out RTS that meets specification and ASC ATDM application requirements
  - Continued engagement with AMT community to see where and how existing RTS could meet DHARMA specification and requirements

Classify runtime design patterns, establish common vocabulary

Sandia National Laboratories

Establish common terminology and best practices with an eye towards eventual standards

Framework	Primary Distributed Memory Model	Primary Shared Memory Model	Supported Distributed Memory Model	Supported Shared Memory Model
Legion	Conservative Disjoint Data	Conservative Disjoint Data	Speculative Copy-on-read	Speculative Atomics
Charm++	Copy-on-read	n/a	n/a	Any
MPI	CSPs	Disjoint Data (OpenMP)	Explicit Sync (One-sided)	Any
UPC	Explicit Synchronization	Explicit Synchronization	n/a	n/a
X10	Conservative Forking	Conservative Forking	n/a	n/a
Cilk	n/a	Conservative Forking	n/a	n/a
CnC	Idempotency	Idempotency	n/a	Any
Chapel	Disjoint Data	Disjoint Data	Any	Any
Uintah	CSPs	Conservative Forking	n/a	n/a
HPX	Data-Flow	Data-Flow	Explicit Synchronization	Explicit Synchronization
TASCEL	Idempotent Data Store	Idempotent Data Store	n/a	n/a
OmpSs	n/a	Conservative Data-Flow	n/a	Explicit Atomics
STAPL	Disjoint Data	Disjoint Data	Explicit Synchronization	Explicit Synchronization
PARSEC	Conservative Data-Flow	Conservative Data-Flow	n/a	n/a

e.g. “Hazard” avoidance models for ensuring correct parallel execution (correct data-flow, no race conditions)

Classify runtime design patterns, establish common vocabulary

Sandia National Laboratories

Engage the community

Establish common terminology and best practices with an eye towards eventual standards

Framework	Primary Distributed Memory Model	Primary Shared Memory Model	Supported Distributed Memory Model	Supported Shared Memory Model
Legion	Conservative Disjoint Data	Conservative Disjoint Data	Speculative Copy-on-read	Speculative Atomics
Charm++	Copy-on-read	n/a	n/a	Any
MPI	CSPs	Disjoint Data (OpenMP)	Explicit Sync (One-sided)	Any
UPC	Explicit Synchronization	Explicit Synchronization	n/a	n/a
X10	Conservative Forking	Conservative Forking	n/a	n/a
Cilk	n/a	Conservative Forking	n/a	n/a
CnC	Idempotency	Idempotency	n/a	Any
Chapel	Disjoint Data	Disjoint Data	Any	Any
Uintah	CSPs	Conservative Forking	n/a	n/a
HPX	Data-Flow	Data-Flow	Explicit Synchronization	Explicit Synchronization
TASCEL	Idempotent Data Store	Idempotent Data Store	n/a	n/a
OmpSs	n/a	Conservative Data-Flow	n/a	Explicit Atomics
STAPL	Disjoint Data	Disjoint Data	Explicit Synchronization	Explicit Synchronization
PARSEC	Conservative Data-Flow	Conservative Data-Flow	n/a	n/a

Starting point for discussion and debate within the community

e.g. “Hazard” avoidance models for ensuring correct parallel execution (correct data-flow, no race conditions)

Sandia's ongoing and future efforts

Gather requirements from a broad set of HPC applications

- Sandia ASC ATDM program focused on gathering requirements for
  - Unstructured, finite element particle-in-cell
  - Reentry application
  - Complex workflows (multi-physics, UQ, in situ analysis)
- Sandia applications deep-dive workshop in July
- Continued engagement with the tools community

Classify application patterns and workloads and how they challenge existing runtime systems

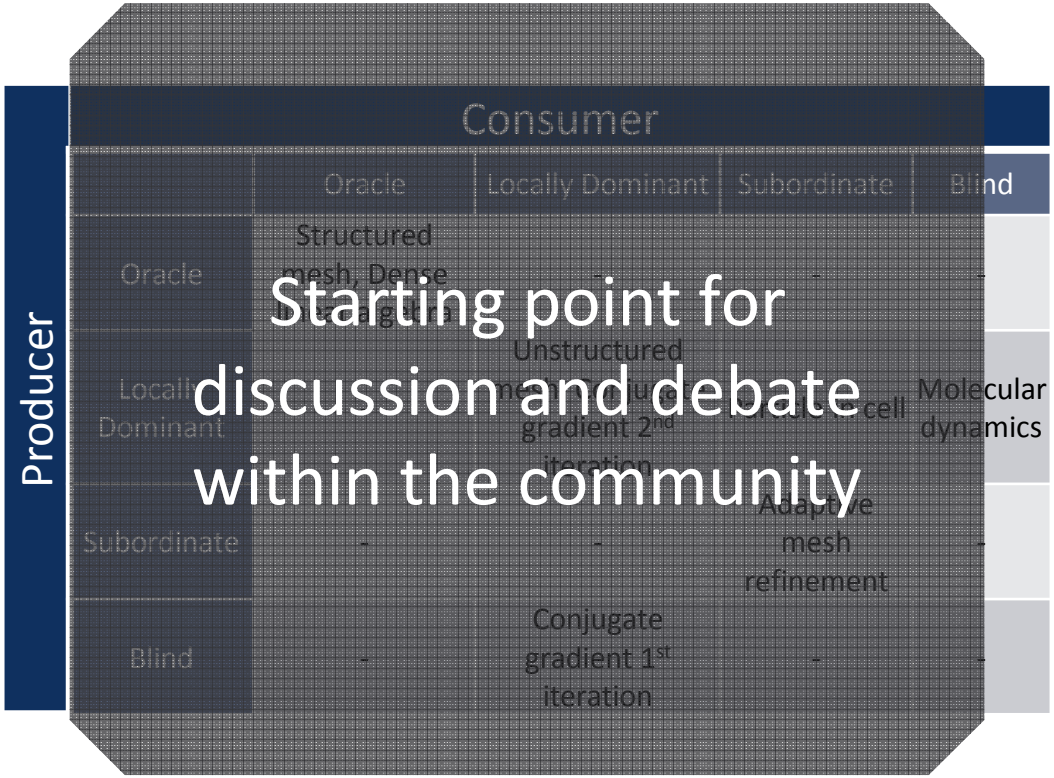
Gather requirements from a broad set of HPC applications

Producer \ Consumer	Oracle	Locally Dominant	Subordinate	Blind
Oracle	Structured mesh, Dense linear algebra	-	-	-
Locally Dominant	-	Unstructured mesh, Conjugate gradient 2nd iteration	Particle in cell	Molecular dynamics
Subordinate	-	-	Adaptive mesh refinement	-
Blind	-	Conjugate gradient 1st iteration	-	-

e.g. Classifying application patterns based on mutually shared knowledge between “producer” and “consumer”

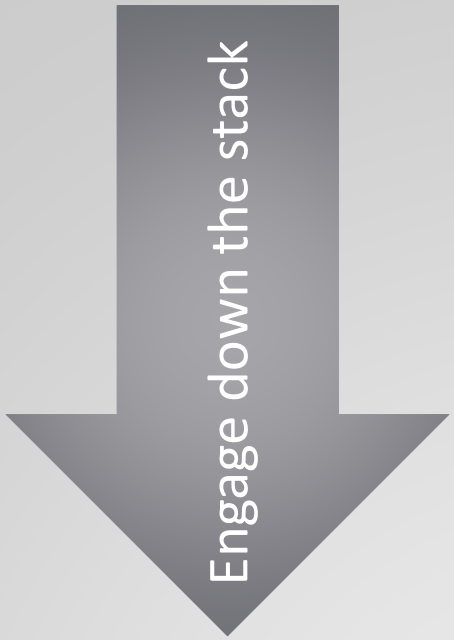
Classify application patterns and workloads and how they challenge existing runtime systems

Gather requirements from a broad set of HPC applications



e.g. Classifying application patterns based on mutually shared knowledge between “producer” and “consumer”

Sandia's ongoing and future efforts



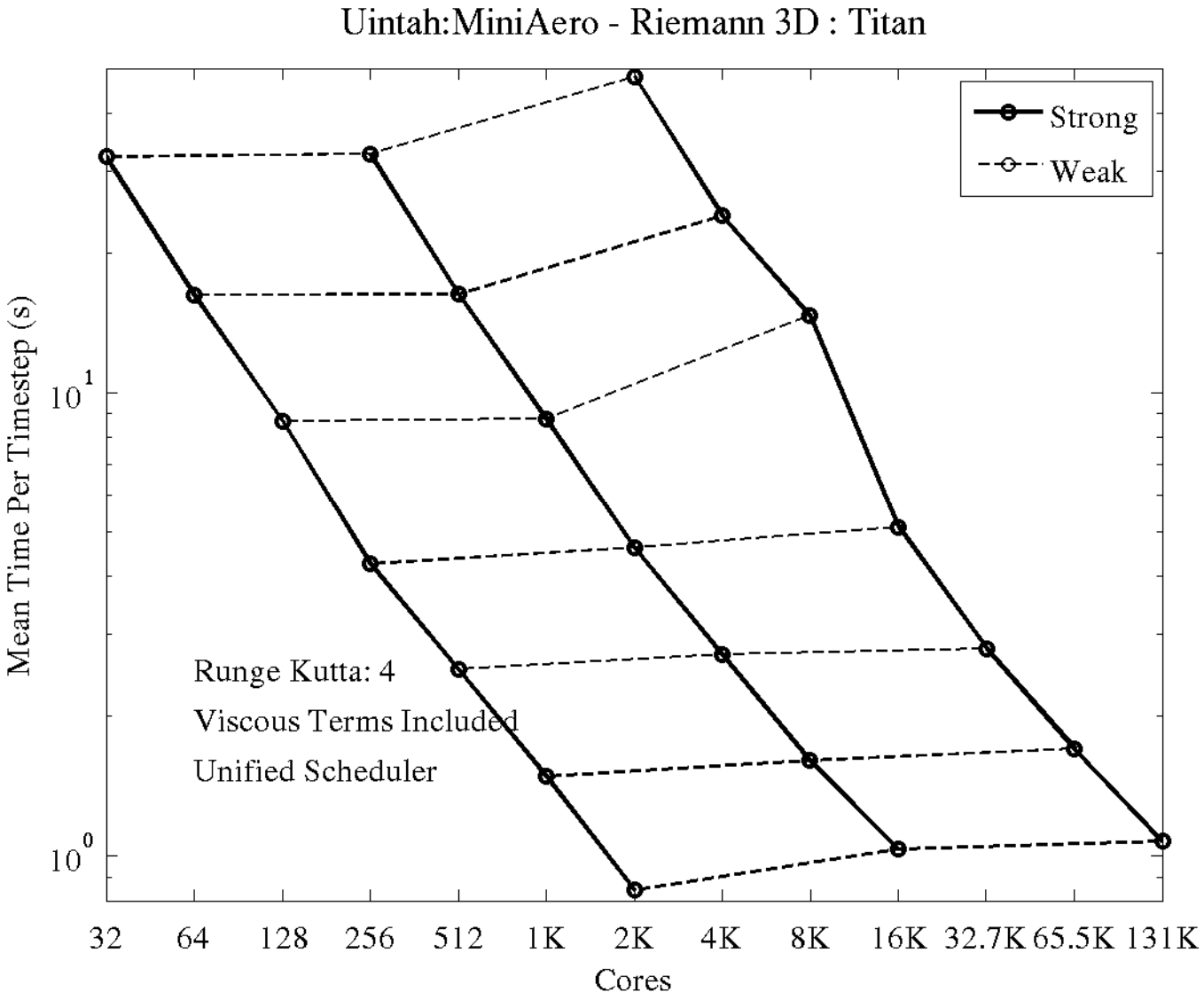
Communicate requirements to OS software developers

- DOE AMT RTS Working Group
  - POCs: Robert Clay and Ron Brightwell
  - Develop pre-standard APIs (not a standards committee!)
  - Augment and support community in efforts to develop sharable AMT RTS components
- Engaging Sandia systems software research groups
  - Qthreads, Kelpie, Kokkos (facilitates statically-defined performance portability)

Extra slides

- Performance analysis slides not ready for prime time – incorporating changes to baseline miniAero and rerunning these

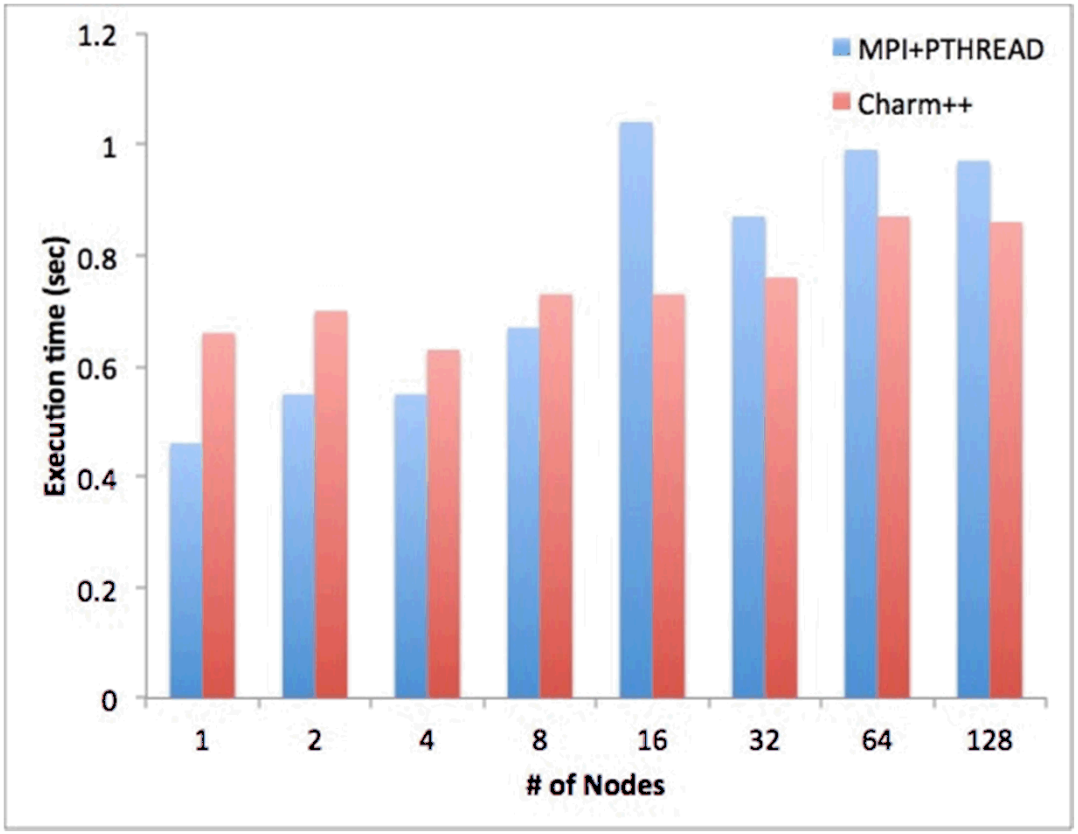
Initial scaling studies show promise for AMTs



Initial scaling studies show promise for AMTs

Sandia National Laboratories

Edison @ NERSC
2 threads spawned per core
No over-decomposition



Both implementations use Kokkos