Sriram Krishnamoorthy
Pacific Northwest National Laboratory

UNIFIED AND INTEROPERABLE PROGRAMMING MODELS

Pacific Northwest
NATIONAL LABORATORY
Proudly Operated by Battelle Since 1965

Programming Models Challenges

- Architectural variability
- Application execution variability
- Algorithmic variability
- Application programmer variability

Architectural Variability

- Concurrency, especially multi-threading
- Memory hierarchy/heterogeneity
- Fault tolerance
- Power/energy consumption

Application Execution Variability

- Strong vs weak scaling
- Fixed point vs dynamics
- Stand-alone or in context of another calculation
- Strongly-coupled vs ensemble/weakly-coupled

Algorithmic variability

- New coupling of existing components
  - Eg., direct vs iterative solutions
- Reformulation of existing algorithms
  - Eg., factorized representation of a specific input operator
- New algorithms
  - Eg., low-order methods with increased sparsity

Application Programmer Variability

- Not all application programmers work at the same level of abstraction
- Black-box/power users
- Developers of calculations/methods
- Infrastructure/runtime developers

Bottomline

- Applications are evolving/adapting
- Architectural trends forcing hardware design choices
- Programming models need to adapt
- Need prog. models
  - Effective application adaptation
  - Allow hardware exploration
  - Engender evaluation and research on all software layers

Evolutionary vs Revolutionary

- Ongoing discussion in the community
  - Recent ASCR workshop on this topic
- Evolutionary approach
  - Adapt existing models for exascale
- Revolutionary approach
  - New languages and models

Designing an Ecosystem

- Collection of programming models
  - Intra-node, inter-node, languages, …
- Inter-operable (and swappable)
- Multiple choices to the programmer
  - Application-customized software stacks
- Performance, portability

Inter-operability

- Application built of many programming models
  - GFMC: MPI + ADLB
  - NWChem: ARMCI + GA + ARMCI + TASCEL
- Composability
  - UPC function calling Co-Array Fortran
  - Progress of MPI, ARMCI, and GASNet calls
- Decomposability
  - Replace C calls with assembly
  - Replace GA calls with ARMCI calls
- Inter-changeability
  - Intel TBB, OpenMP
  - C, C++, and Fortran

Composition: Beyond Inter-operation

func(A,B)	Can cached values be reused?	Object attributes
C = A op B	Is owner-computes load balanced? Is there sufficient parallelism?	Profile-guided parallelization & scheduling
C = A op B; D = C op E	Does all of C need to be computed before it can be used?	Consistency properties; producer-consumer pipelining
while (i++) func();	What information from the previous execution of func() is still valid?	Conditional profiles

Sequential Ecosystems

Common Language Runtime (.NET)	Java Virtual Machine (JVM)	Low-level Virtual Machine (LLVM)
C#, F#, VB.NET, J#, IronPython, C++/CLI, …	Erjang, Rhino, Jython, Free Pascal, Jruby, … MIDletPascal, Clojure, Groovy, Scala, Fantom, BBj	C, C++, Objective-C, Fortran, Python, Ruby, Java bytecode, …

Our Ecosystem

Domain-specific languages

Performance tools

Application runtimes

Parallel compilers

Debugging tools

Task-based libraries

Math libraries

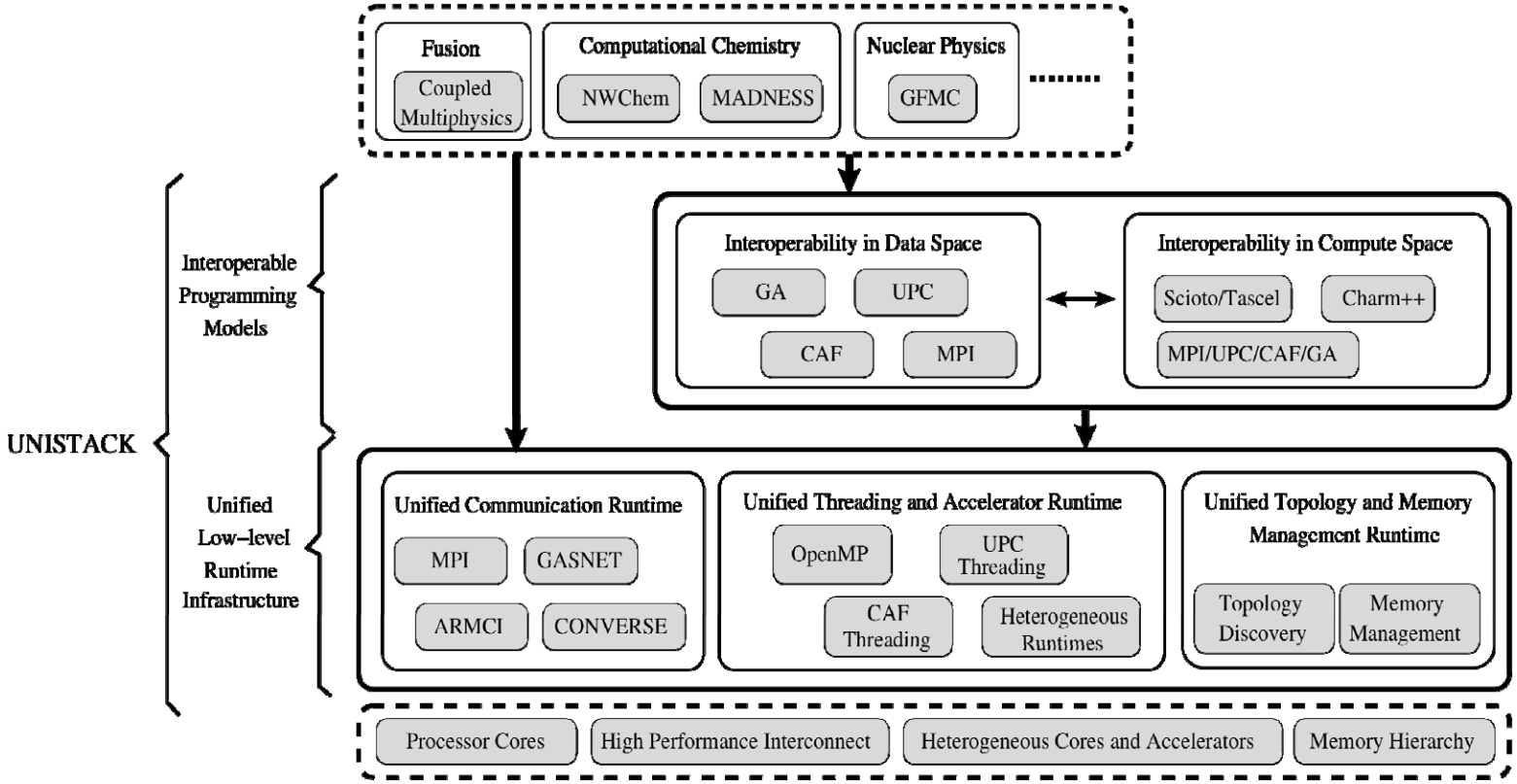
Threading languages/runtimes

Communication libraries

Sequential compilers

Assemblers

A Solution: UNISTACK



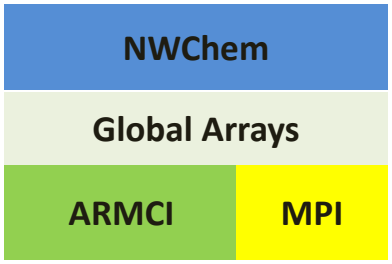
- Unified software stack
- Subset of programming models
- Demonstrable impact on applications

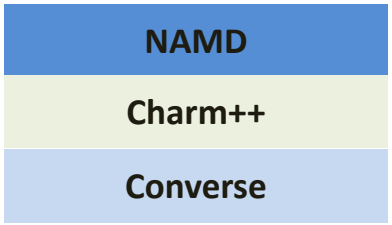
Low-level Communication Libraries

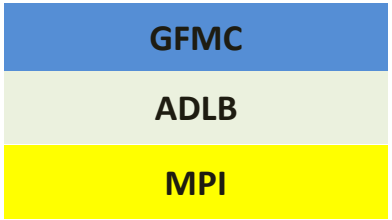
- MPI, ARMCI, GASNet, Converse

- Two-sided messaging
- Remote memory access
- Active messages
- Collective communication

- Communication progress
- Shared helper thread
- Memory registration caches
- Buffers, connections, …







Intra-node Primitives

- OpenMP, Charm++ runtime, …
- Thread management
- Affinity domains
- Atomcity/locking model
- Scheduling

Data Space Models

- Data allocated by : UPC, Global Arrays, Co-Array Fortran, …
- Accessed by: any model
- Data structure handles and structure information
- Synthesize views of data allocated by another mode
- Consistency semantics

Compute Space Models

- Process-centric
- Inter-node task/data parallelism
- Locality-directed computation

Application Impact

- Improved performance and resource utilization
- Incremental optimization/porting
- Mix thread scheduling, data structures, and load balancing components from different models
  - “Shop” for components wherever they can be found
  - Selectively develop what is missing

Desired Outcomes

- Unify state-of-the-art research
- Identify gaps
  - Is a “better” implementation possible outside the stack?
  - How can we include that?
  - Eg: ARMCI influencing MPI 3.0 RMA standard
- Fertile substrate for research
  - Custom data structures; small changes to runtimes; …

Beyond Engineering

- Hard problems: deadlock freedom; language-runtime specification; consistency semantics; …
- How many implementations of quick sort?
  - We don't take away people's toys
  - Focus on the critical problems
- Spur cross-cutting research
  - Across programming models
  - Prog. models and applications