(De)composable Abstractions for a Changing Architectural Landscape

Sriram Krishnamoorthy

Pacific Northwest National Lab

ASCR Programming Challenges Workshop
July 2011

Pacific Northwest
NATIONAL LABORATORY

Proudly Operated by Battelle *Since 1965*

Programming Model 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

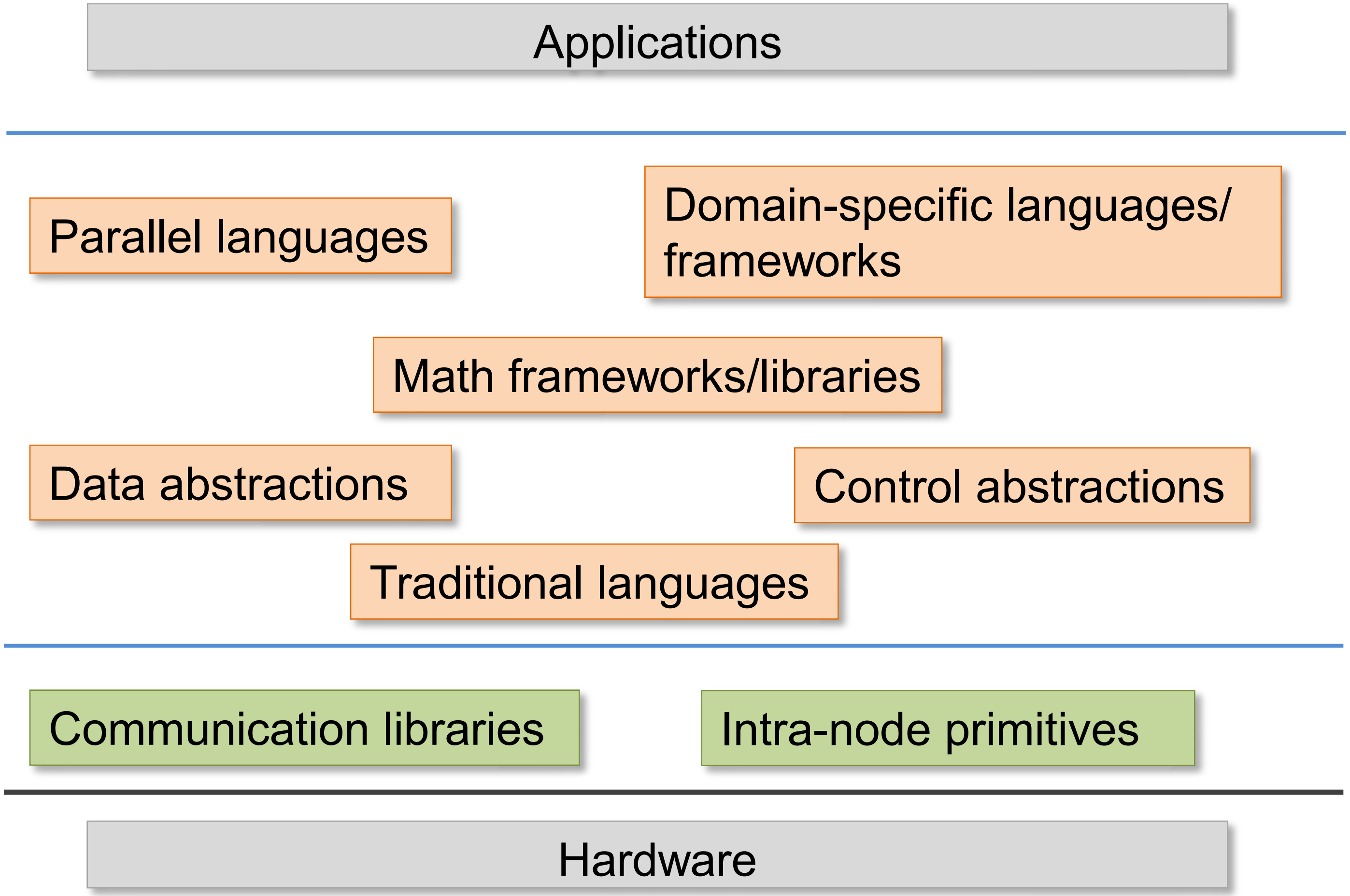
- Not all application programmers work at the same level of abstraction
- Black-box/power users
- Developers of calculations/methods
- Infrastructure/runtime developers

The Objective

> “Premature optimization is the root of all evil.”
> -- Donald Knuth

- Sustainable abstractions
  - Maintainable over the next decade(s)
- Accessible to domain experts
  - Encode today’s and tomorrow’s algorithms
- Flexible and optimizable
  - Handle real application scenarios
  - Enough information for compile-time/runtime optimization

Programming Model Ecosystem



Applications

Parallel languages

Domain-specific languages/ frameworks

Math frameworks/libraries

Data abstractions

Control abstractions

Traditional languages

Communication libraries

Intra-node primitives

Hardware

We may not replace all modules… for all applications … right now

Evolutionary Approach

> “The competent programmer is fully aware of the strictly limited size of his own skull; therefore he approaches the programming task in full humility, and among other things he avoids clever tricks like the plague.”
>
> -- Edsger W. Dijkstra

- Collection of inter-operable models
- Composable abstractions
- Decomposable abstractions
- Auto-tune and generate code where possible

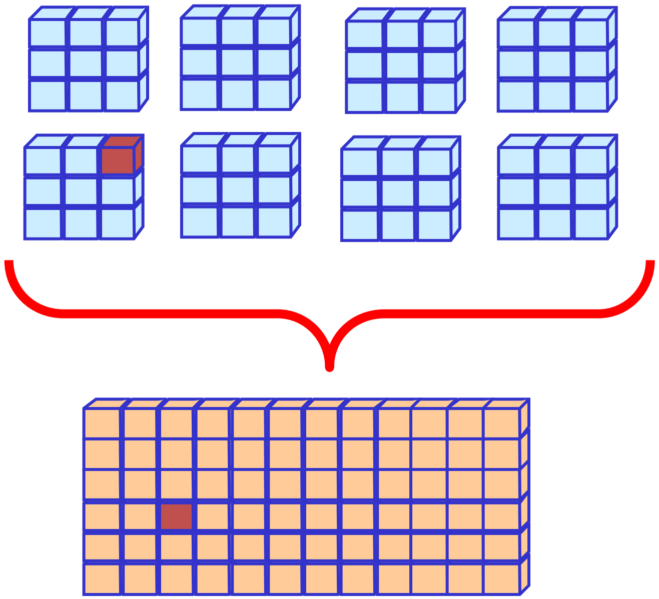
Collection of Inter-operable Models

- Partitioned global address space data
  - Inter-operable with MPI
- Task-based execution model
  - Iterative and recursive parallelism
- Phase-based execution: Switch between
  - SPMD and task-based execution modes
  - GAS and partitioned data views

Partitioned Global Address Space Data

- Exposes application data structures to runtime
- Data locality exposed to the user and runtime
- Communication operations visible in the program
- High-level operations on global data
- Scoped direct access to local data

Physically distributed data



Global Address Space

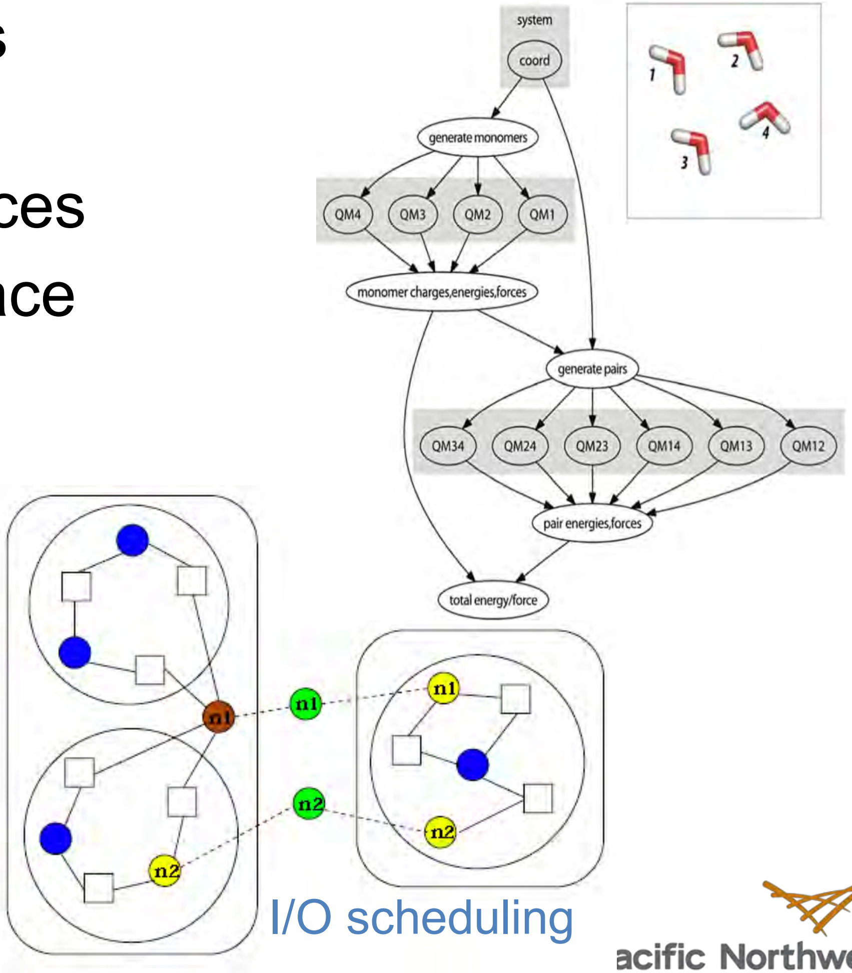
Pacific Northwest
NATIONAL LABORATORY

Proudly Operated by Battelle *Since 1965*

Task-based Execution Model

- Work as collection of tasks
  - Over-parallelize
- Specification of dependences
- Data in global address space
  - Enables task migration
- Building blocks for
  - Functional models
  - Task-graph scheduling
  - Work stealing
  - …

Mixed QM/MM calculation



I/O scheduling

Composing Abstractions

- Optimized implementations of individual operations
  - Potentially in different prog. models

- Can we combine them effectively?

- Translate domain information into runtime attributes
  - User provided
  - Runtime inferred

- Intelligent and adaptive runtime

Composing Abstractions: Elements

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

What transformations can be performed with this information?

Decomposable Abstractions

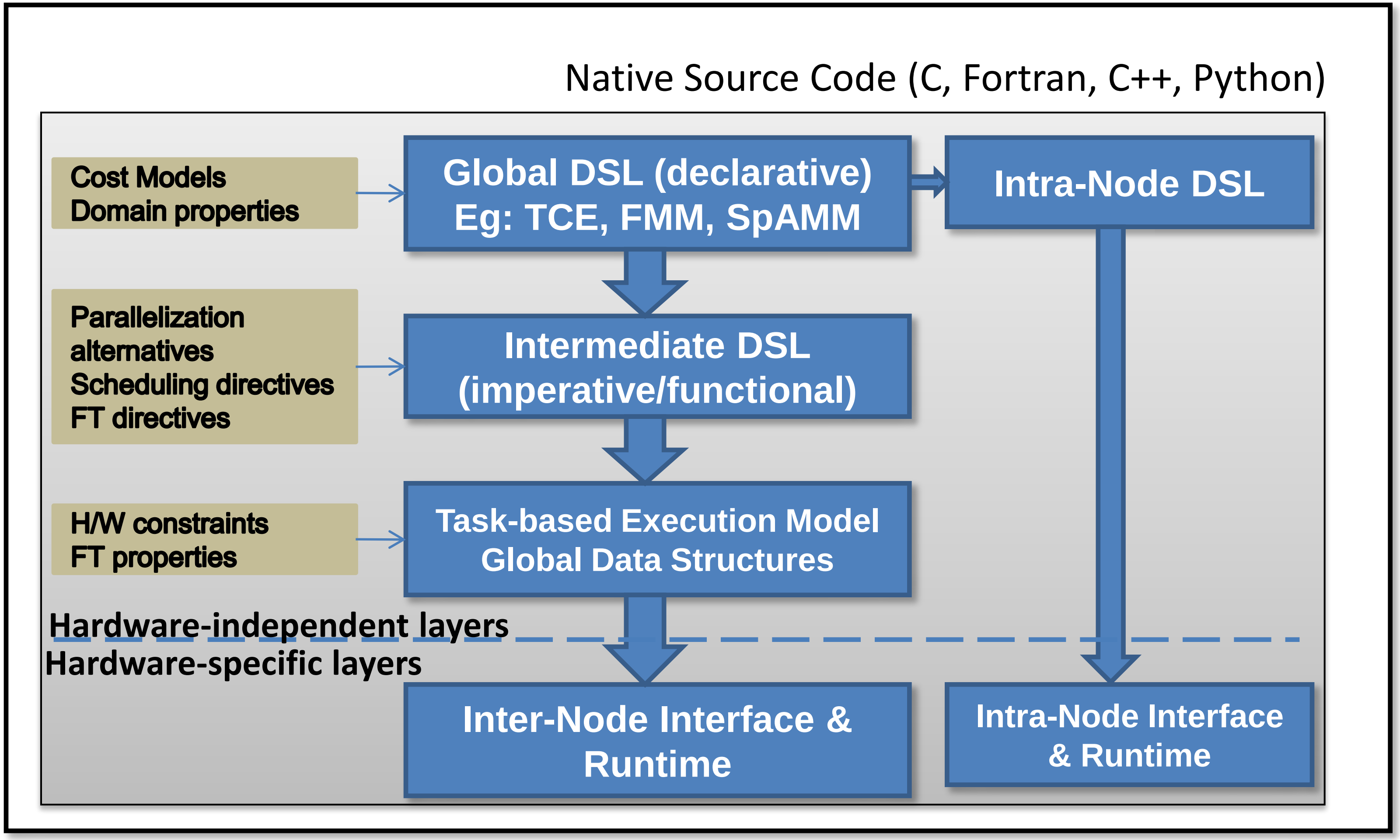
> “The complexity of software is an essential property, not an accidental one. Hence, descriptions of a software entity that abstract away its complexity often abstracts away its essence.”
>
> -- Fred Brooks

- What if the provided abstraction does not suffice?
  - Expressivity
  - Performance

- Provide a lower level of abstraction
  - But still in the eco-system

- Implementation still as readable and adaptable

Decomposable Abstractions : Illustration



Pacific Northwest
NATIONAL LABORATORY

Proudly Operated by Battelle *Since 1965*

Transitioning Users

- Embedded DSLs/directives that get transformed
  - Automatic injection
- Phase-based execution
  - Incremental injection of new programming models
- Accessible transformations to intermediate models
  - Let users change the decisions made
- Software inter-operability

Reality Check

> "… generalizations are like spectacles for the ~~short-sighted~~ blind. They help, but they are no substitute for excellent eyesight, ..."
> -- Bernard Cafferty

- Benchmarks help, but cannot replace application understanding
- Success metric: Application scientists using the programming model
- Not all domain experts work at the same level of abstraction
  - But they share our concerns and are motivated to look for solutions

Be revolutionary, but take the users along!