SLEEC: Semantics-rich Libraries for Effective Exascale Computation

Milind Kulkarni, Arun Prakash, Vijay Pai and Sam Midkiff

Michael Parks

PURDUE UNIVERSITY

Sandia National Laboratories

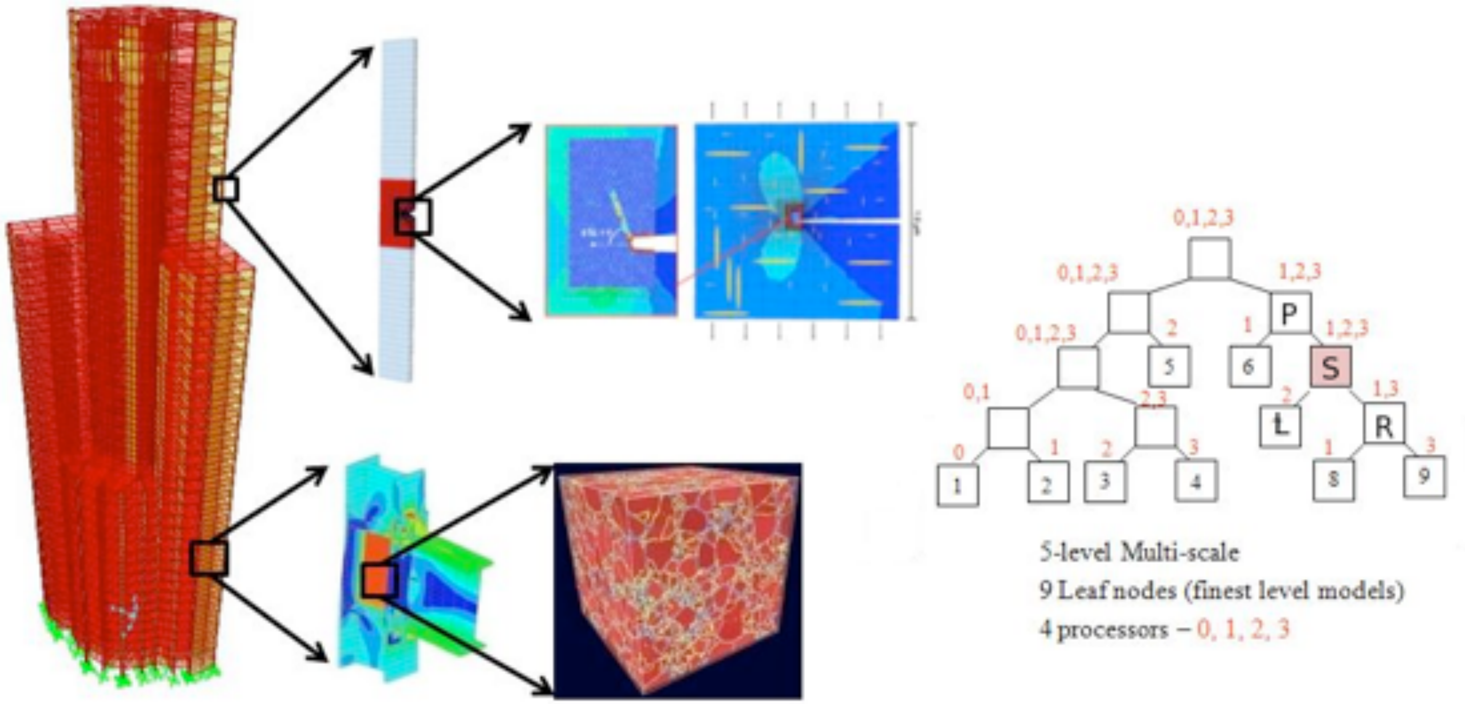
https://engineering.purdue.edu/SLEEC

Motivation

- Modern computational science applications composed of many different libraries
  - Computational libraries, communication libraries, data structure libraries, etc.
  - Peridigm, developed by Mike Parks, builds on 10 different Trilinos libraries
- Each library has its own idioms and expected usage
- Determining right way to compose and use libraries to solve a problem is difficult

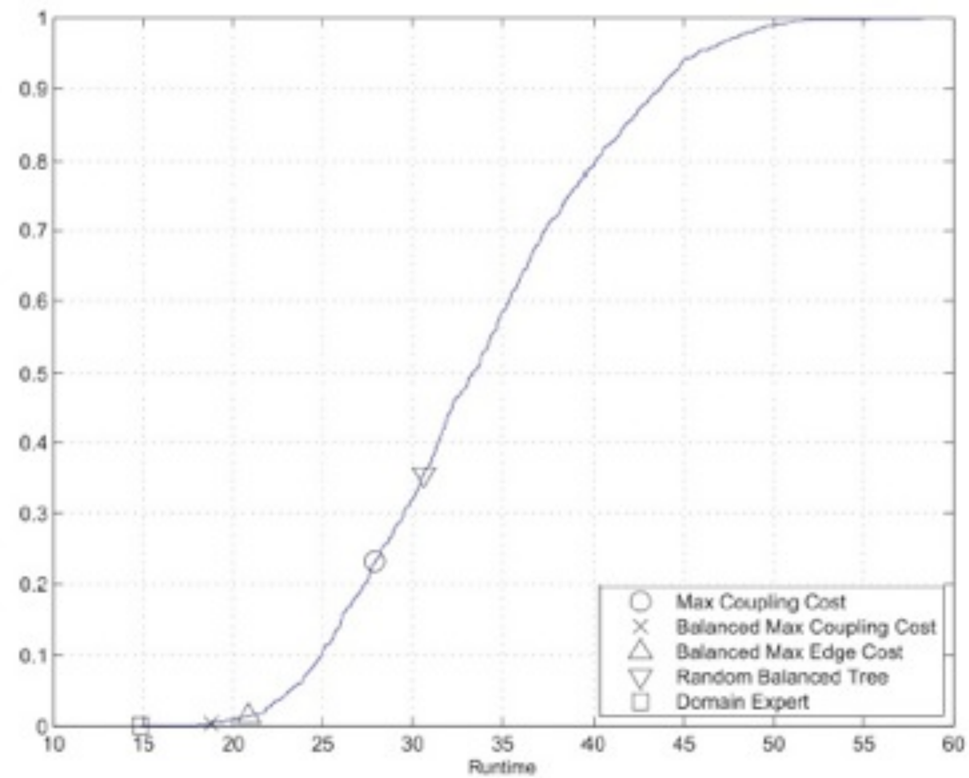
Motivation: Compositional complexity

- Consider loosely-coupled multi-scale computational mechanics problem (developed by co-PI Arun Prakash)
- Must determine right way to decompose problem, couple separate solutions, etc.



Motivation: Compositional complexity

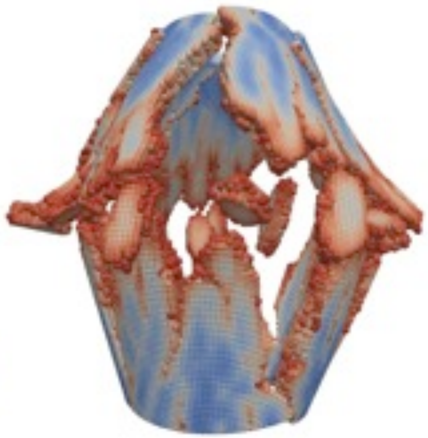
- Simple case: fixed number of subdomains, only consider how to couple them together
- Vast space of configurations: 8 subdomains → 135K possible schedules
- Large variation in performance of different orders
- Exploration of different variants requires knowledge of domain semantics, cost estimates



Motivation: Difficult interaction between libraries

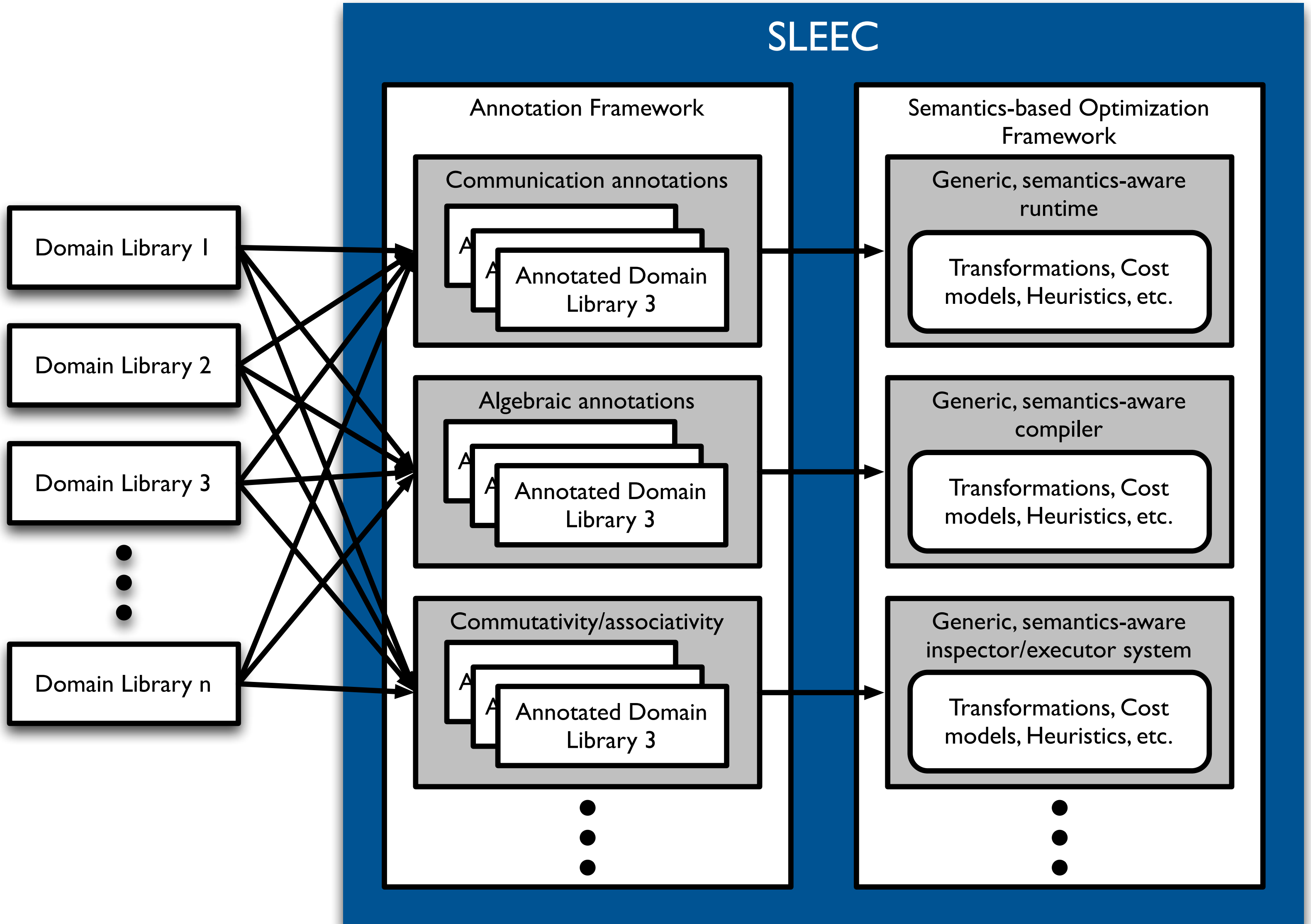
- Peridigm: computational peridynamics code
  - Allows modeling of materials under stress without explicit accounting for discontinuities (fractures, etc.)
- Built on Trilinos components
  - Set of computation and communication libraries
- Requires careful coordination of data movement operations to manage shadow data, etc. needed by solvers
  - But data movement requirements can be directly inferred from which equations are being solved

Before

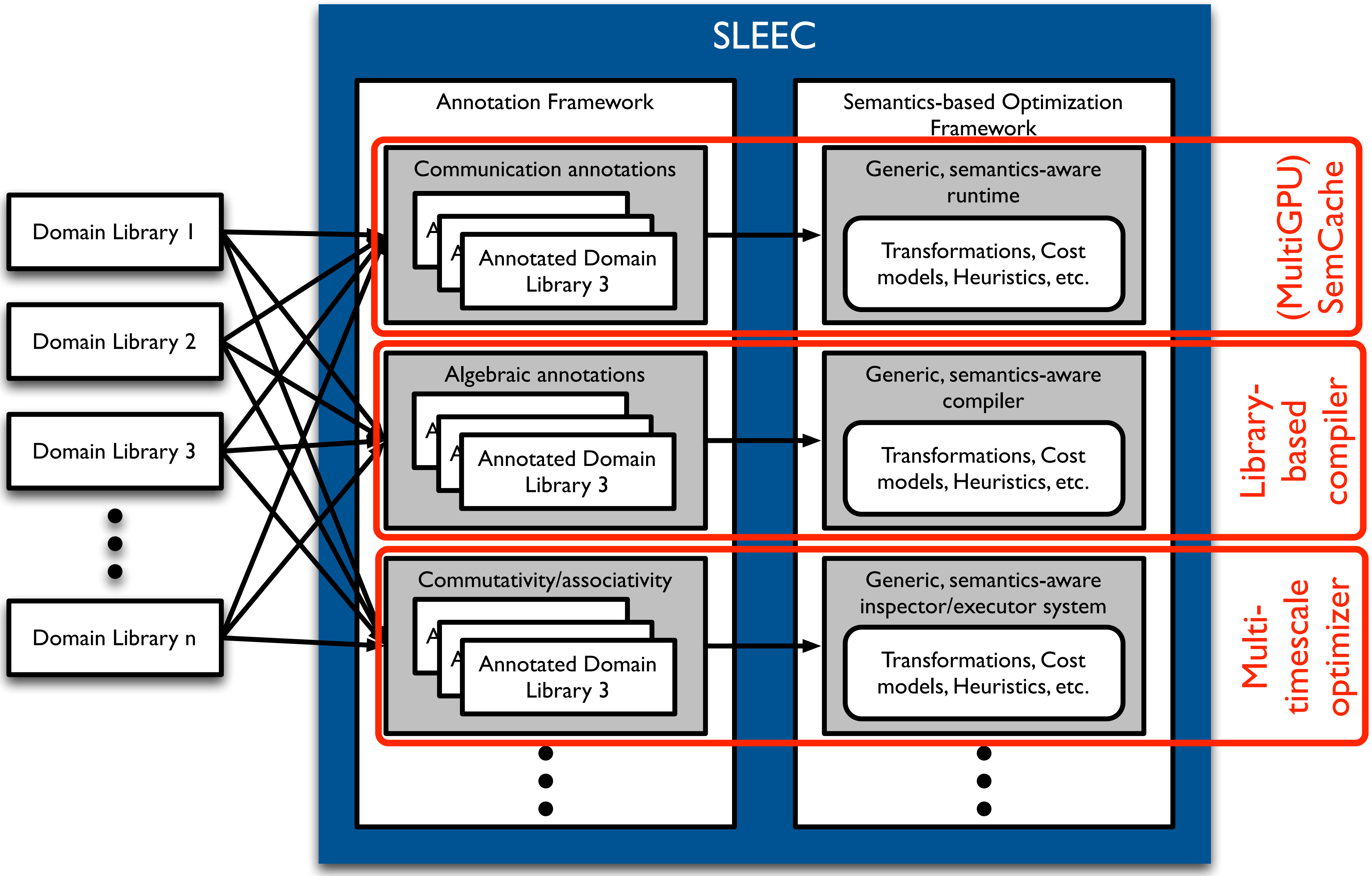


After

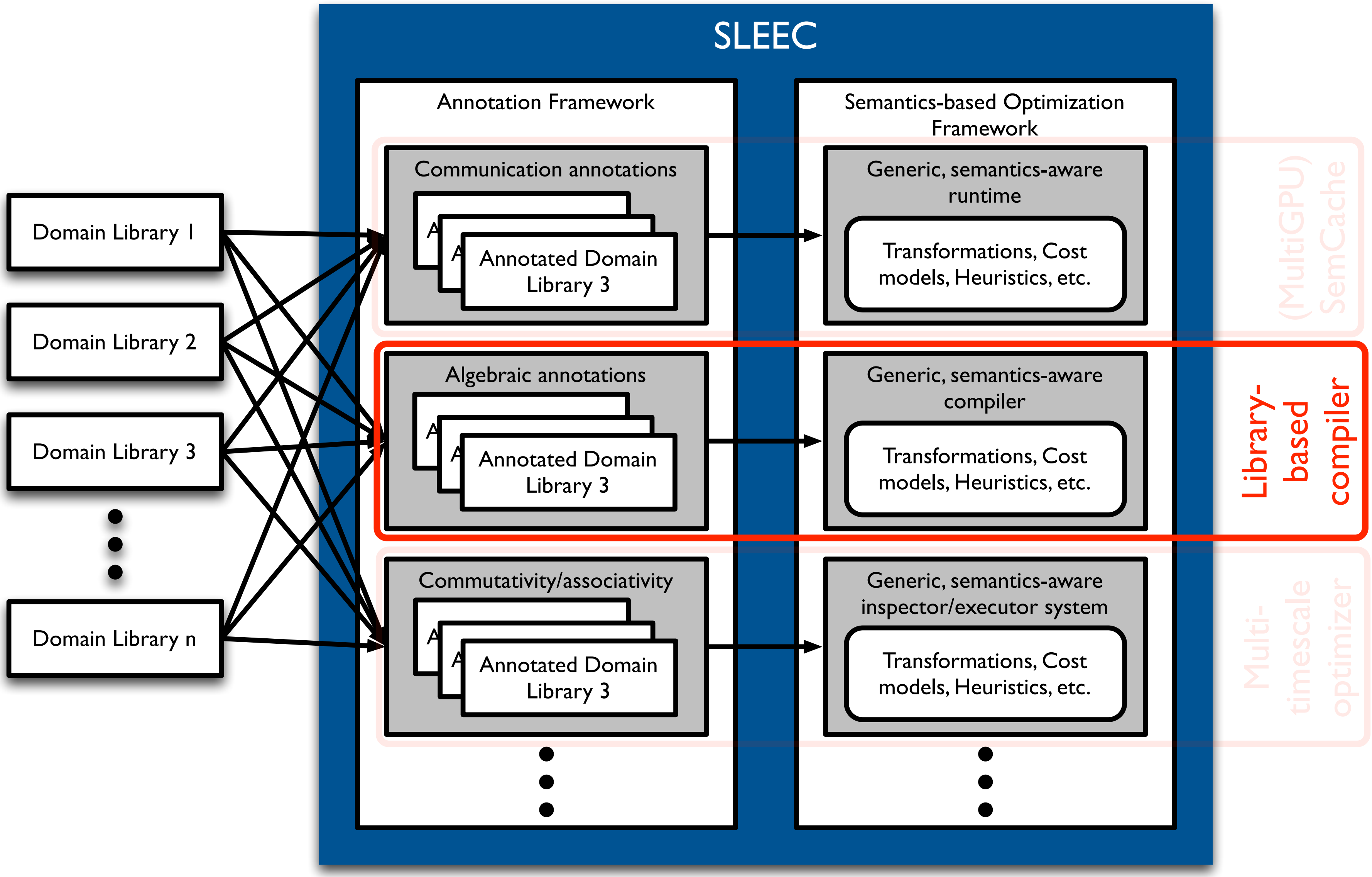
Project vision



Project status



Semantics-based compilation

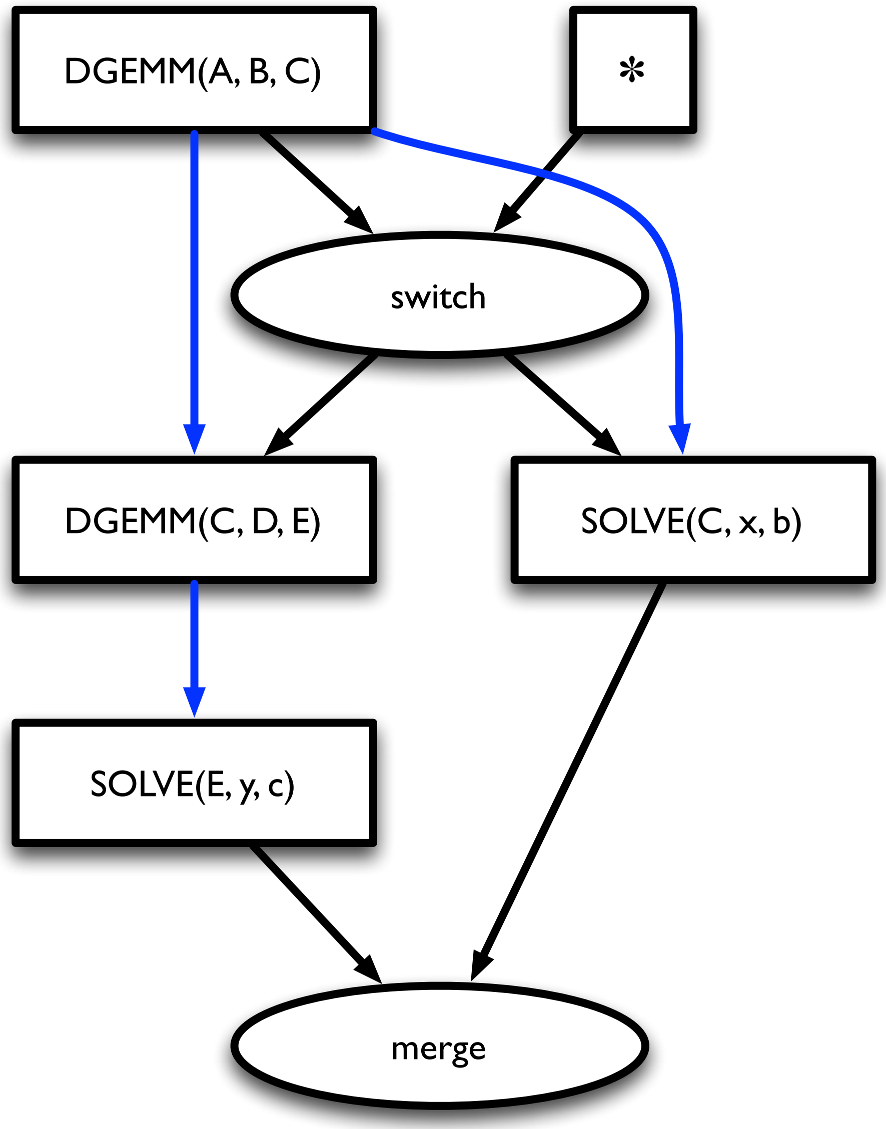


Developing intermediate representation

- Building intermediate representation based on dependence flow graphs
  - Library calls represented as single operations in graph
  - Directly captures dependences between operations
  - Directly represents control flow information (vs. "sea of nodes" IRs) – facilitates re-generating high-level code

IR example

```
DGEMM(A, B, C) //C = A * B
if ( * )
    SOLVE(C, x, b) //solve Cx = b
else
    DGEMM(C, D, E) //E = C * D
    SOLVE(E, y, c) //solve Ey = c
```

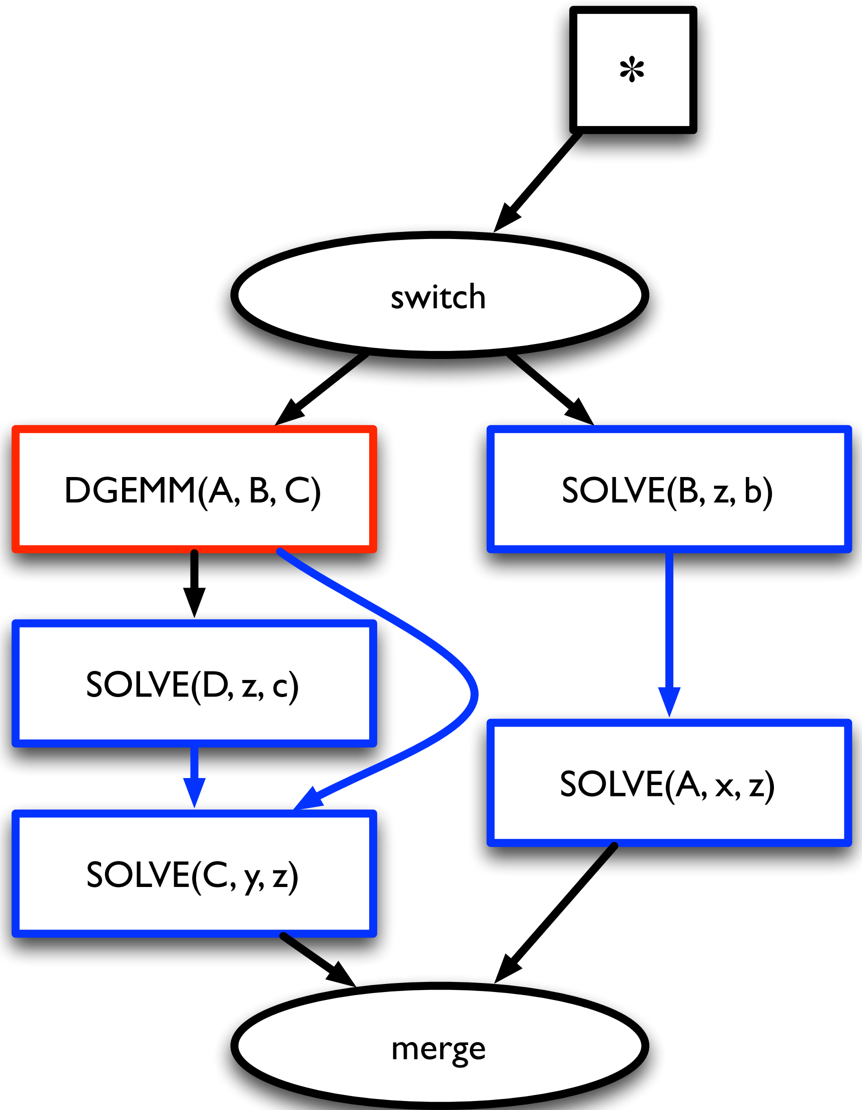


Semantics-based transformations

- Can identify opportunities for transformations based on dependence structure of code
  - e.g., turning multiply followed by solve into two solves
- Some transformations may not be possible due to multiple uses of results of methods
  - When possible, will replicate calls (without introducing redundancy) to facilitate extra transformation

Transformed code

```
if ( * )
    SOLVE(B, z, b)
    SOLVE(A, x, z)
else
    DGEMM(A, B, C) //C = A * B
    SOLVE(D, z, c)
    SOLVE(C, y, z)
```



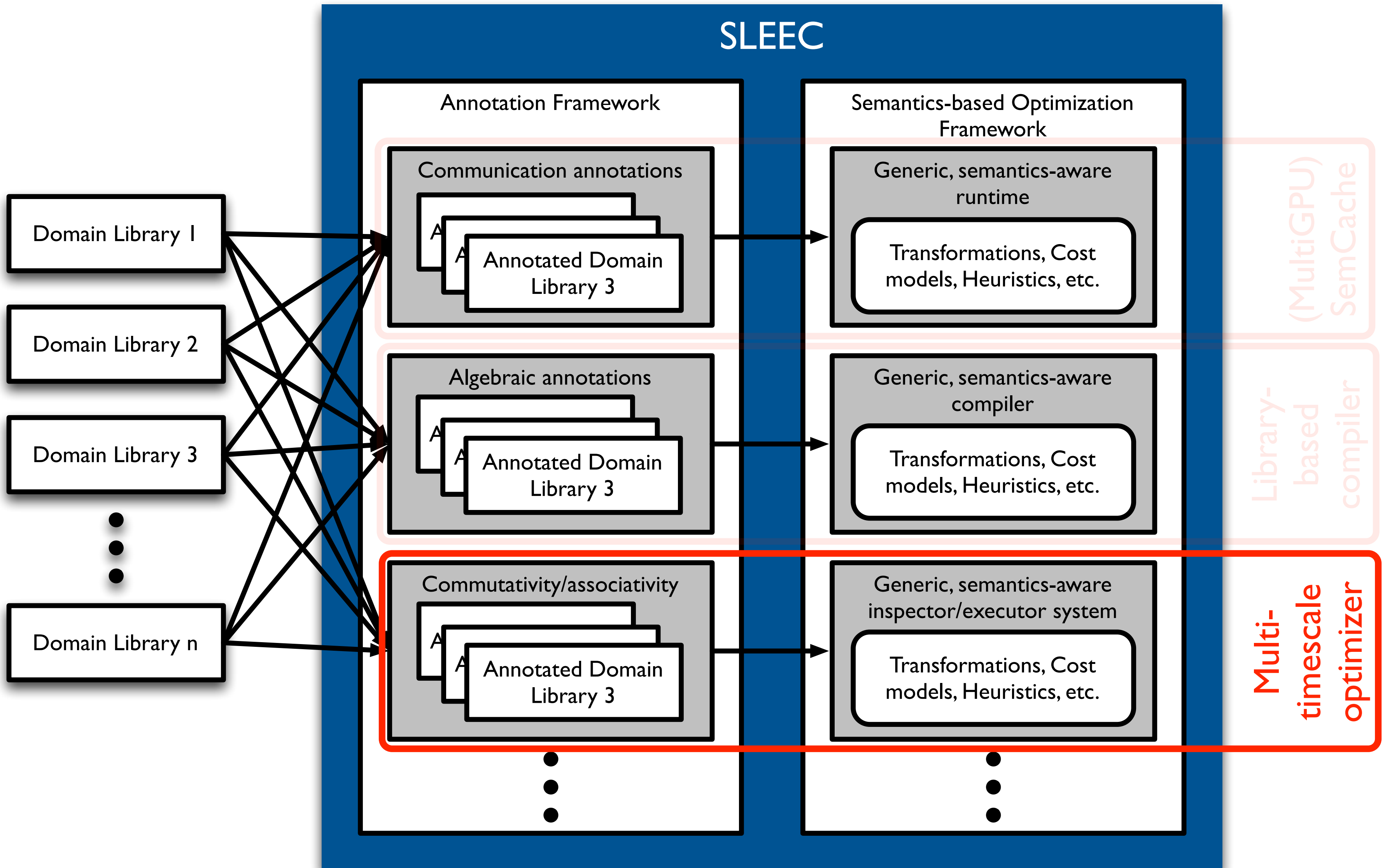
More on transformation framework

- Performs type inference for matrix types
  - Tracks whether matrices are triangular, etc.
  - Allows specialization of functions (replace general solve with triangular solve)
  - Allows cost-model-driven transformation (two solves over triangular matrices faster than multiplying them together then solving)
- Prototype works for subset of BLAS
- Paper under preparation

Integration with ROSE infrastructure

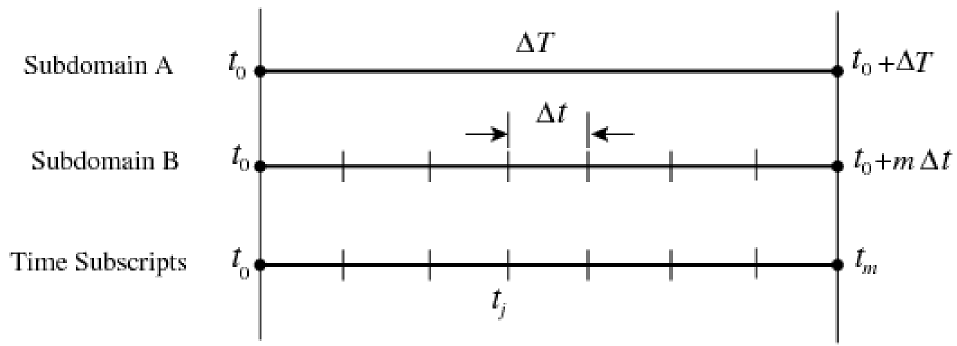
- Current IR built in ROSE
- Analysis and transformations built using *ad hoc* framework
- SLEEC student, Jad Hbeika, going to LLNL this summer to work with Greg Bronevetsky
- Will extend Fuse (extensible ROSE analysis framework) to work with complex data types such as matrices/submatrices
  - Provide enhanced analysis capabilities to applications that ROSE can compile
- Will adapt our transformation framework to work with Fuse

Multi-timescale optimizer



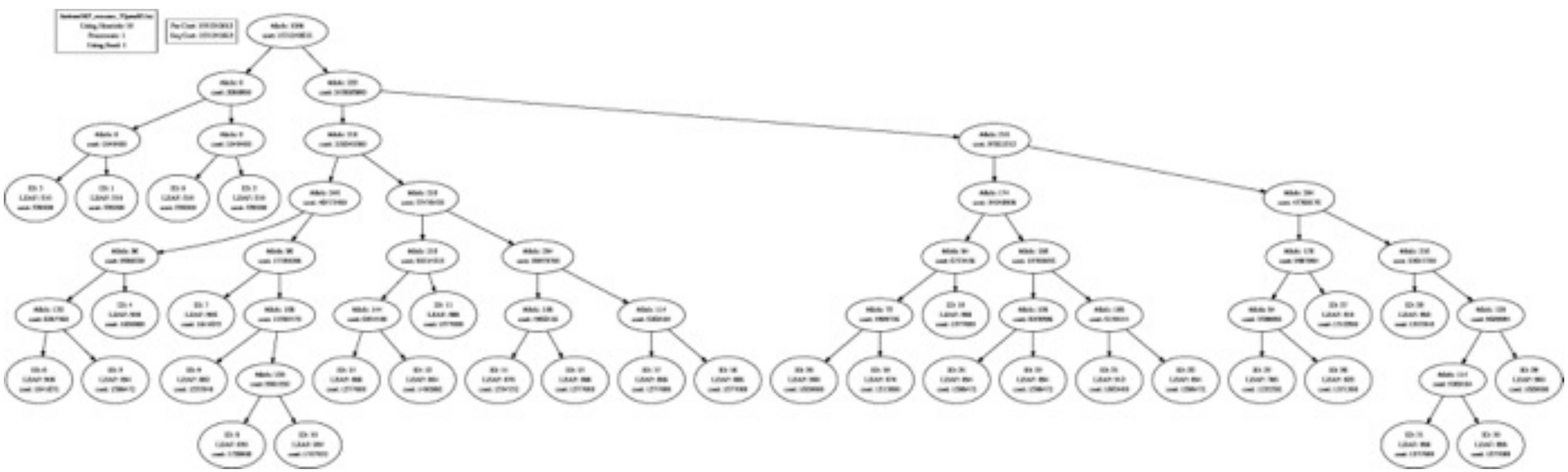
Computational mechanics

- Target: multi-scale computational mechanics codes
  - Loosely coupled problem as in intro
  - Different subdomains use different time steps (smaller time steps for subdomains that need more accuracy)



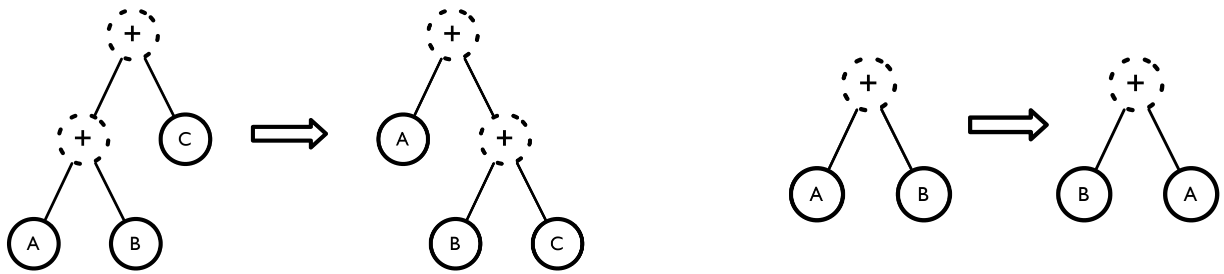
Coupling trees

- Two basic operations:
  - `LeafSolve`: solve a single subdomain at a given time step
  - `Couple`: merge solutions from two subdomains to form "larger" subdomain



Optimizing coupling trees

- `Couple` is associative and commutative



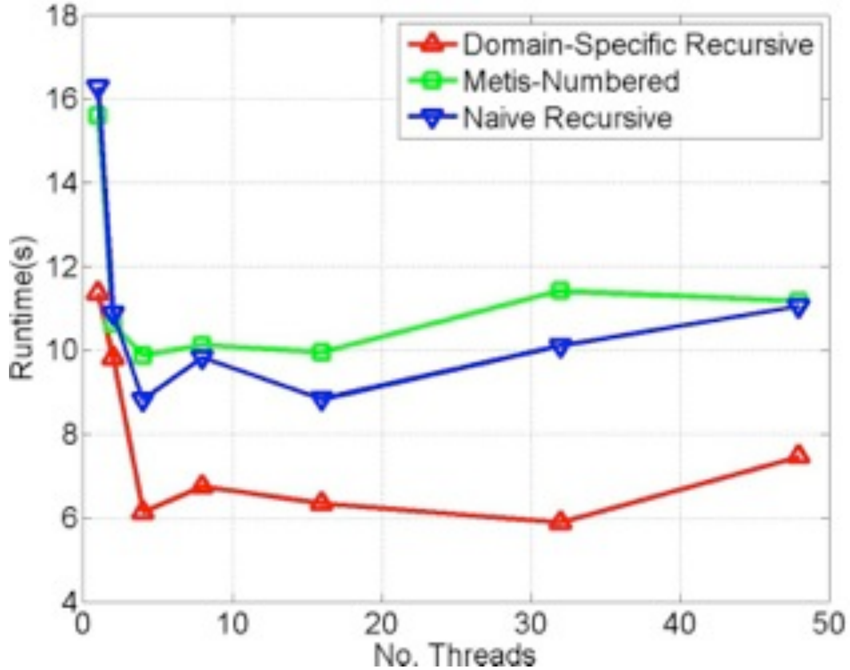
- `Couple`'s operands are also independent (parallelizable)
- Additional restriction based on domain: all domains at a given time step must be coupled before coupling with domains at other time steps
- Can be integrated into basic transformation rules:
  - Each operand has time step information
  - Time step of `Couple(a, b)` result is max(a, b)
  - `Couple` only associative if all operands are at the same time step

Optimizing coupling trees

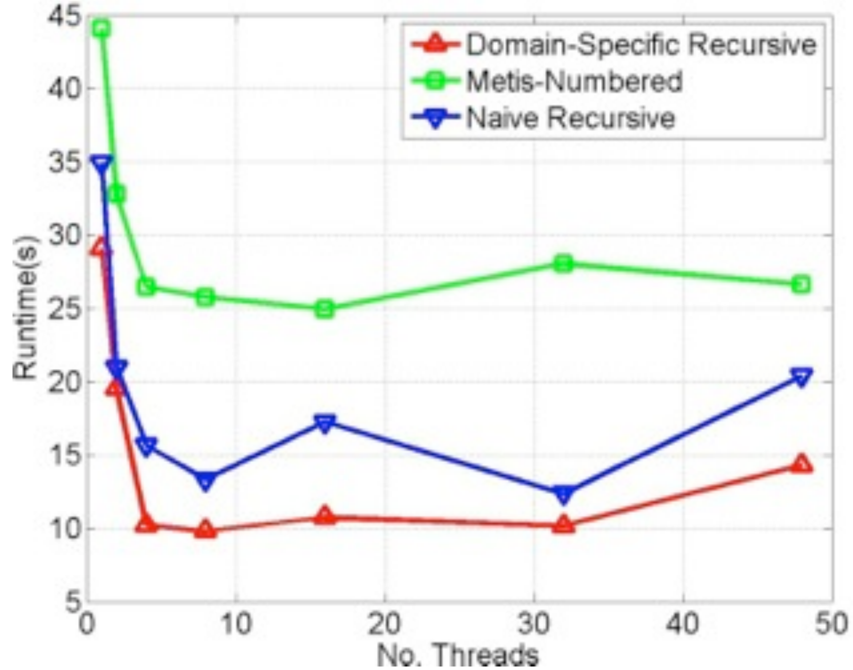
- Cost models for LeafSolve and Couple
  - LeafSolve: based on size of subdomain
  - Couple: based on size of interface between coupled subdomains, and time step ratio of subdomains
- Built heuristic based on costs
  - Attempts to produce balanced trees while minimizing overall cost and respecting constraints on coupling

Results

- Compared to two other variants:
  - "Metis-numbered" – the initial tree order provided by the application writer
  - "Naive recursive" – using the same scheduling heuristic and constraints without taking into account timestep-based cost models



cube



stargrain

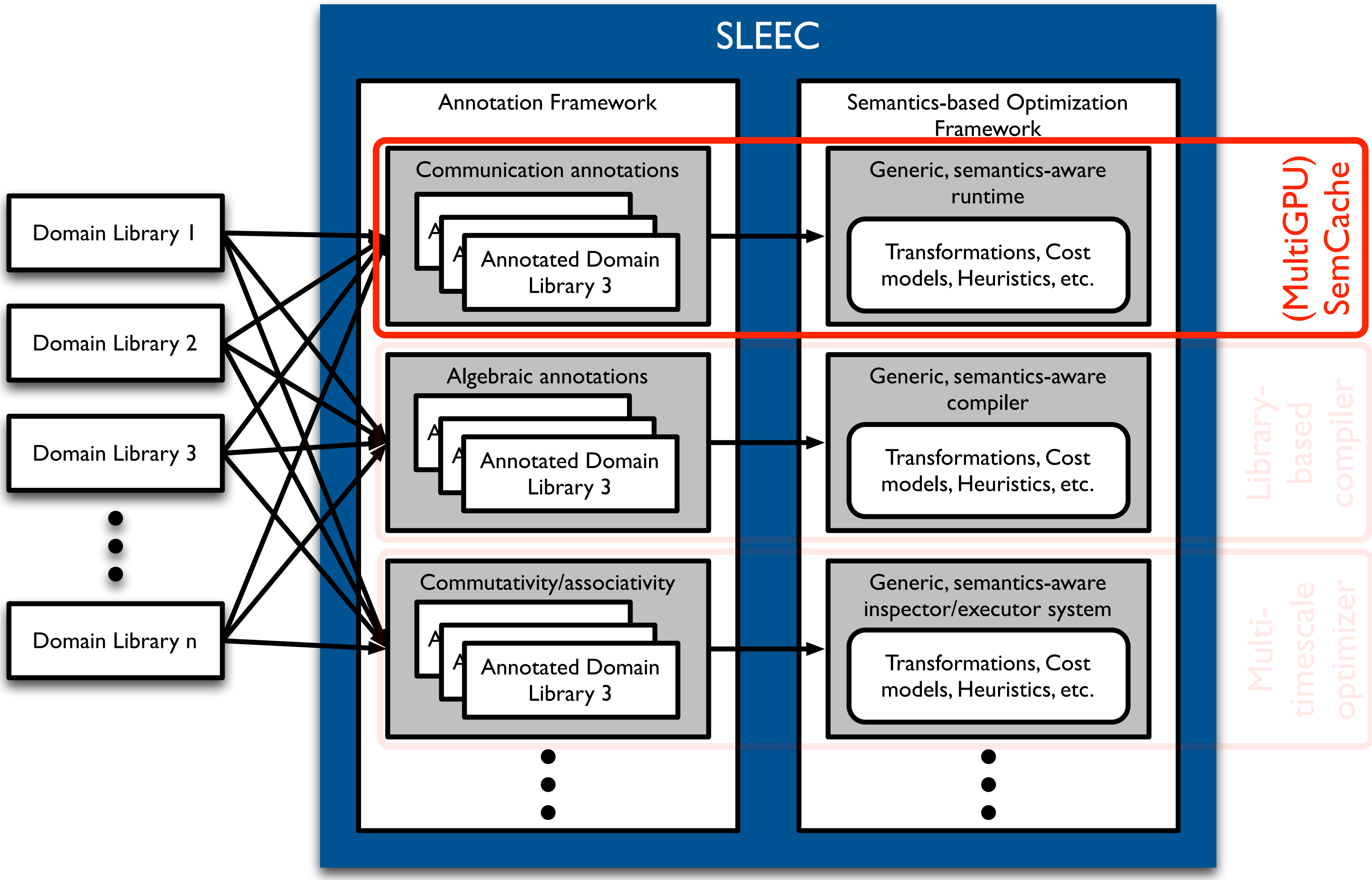
Extension to other domains

- SLEEC student, Payton Lindsay, has been collaborating with PI Mike Parks to develop multi-timescale version of Peridigm
  - Key challenge: "interface" between domains in peridynamics very different for interface in computational mechanics
  - Paper under preparation

Use case: Cross-domain application of semantics-based infrastructure

- Peridynamics has different operations than computational mechanics, but have same high level semantics
  - Recall two basic operations: "solve" a subdomain and "couple" two subdomains
  - Solving a subdomain = solving peridynamics problem
  - Coupling subdomains = exchanging information at boundary layer, which extends *into* each subdomain
- But coupling is still associative and commutative
- Can directly apply scheduling framework, as framework does not care about concrete operations, but only high level semantics

Optimizing communication/synchronization for accelerators



GPU offloading

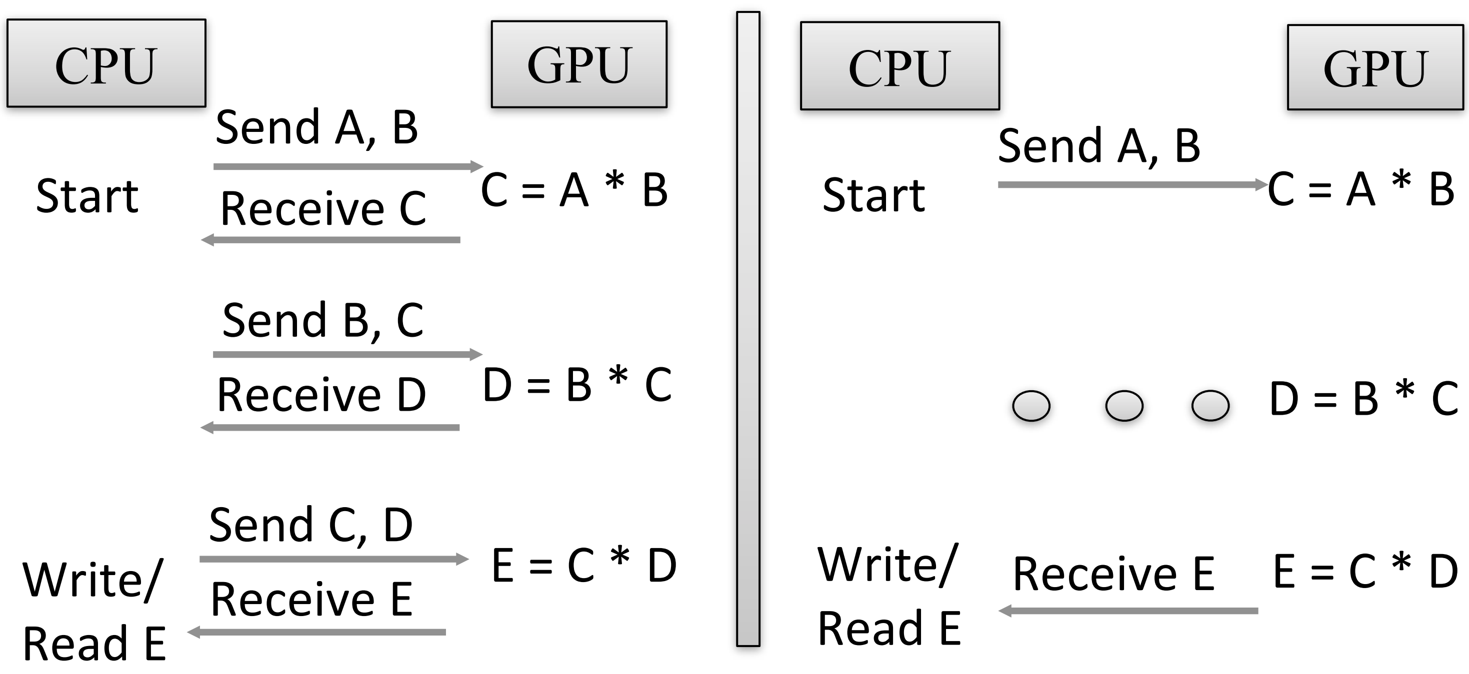
- One approach to heterogeneous computing: *offload* computationally-intensive libraries to GPU
- Advantages
  - Easy to program (just replace library calls!)
- Disadvantages
  - No notion of how library calls interact
- Existing library-based approaches either
  - Take control of all communication, introducing overhead (CULA)
  - Leave communication up to the programmer, losing programmability (Cublas)

Example

1. *BLAS(A x B = C); //matrix multiply*
2. *BLAS(B x C = D); //matrix multiply*
3. *BLAS(C x D = E); //matrix multiply*

(a) Communication un-optimized

(b) Communication optimized



What are my options?

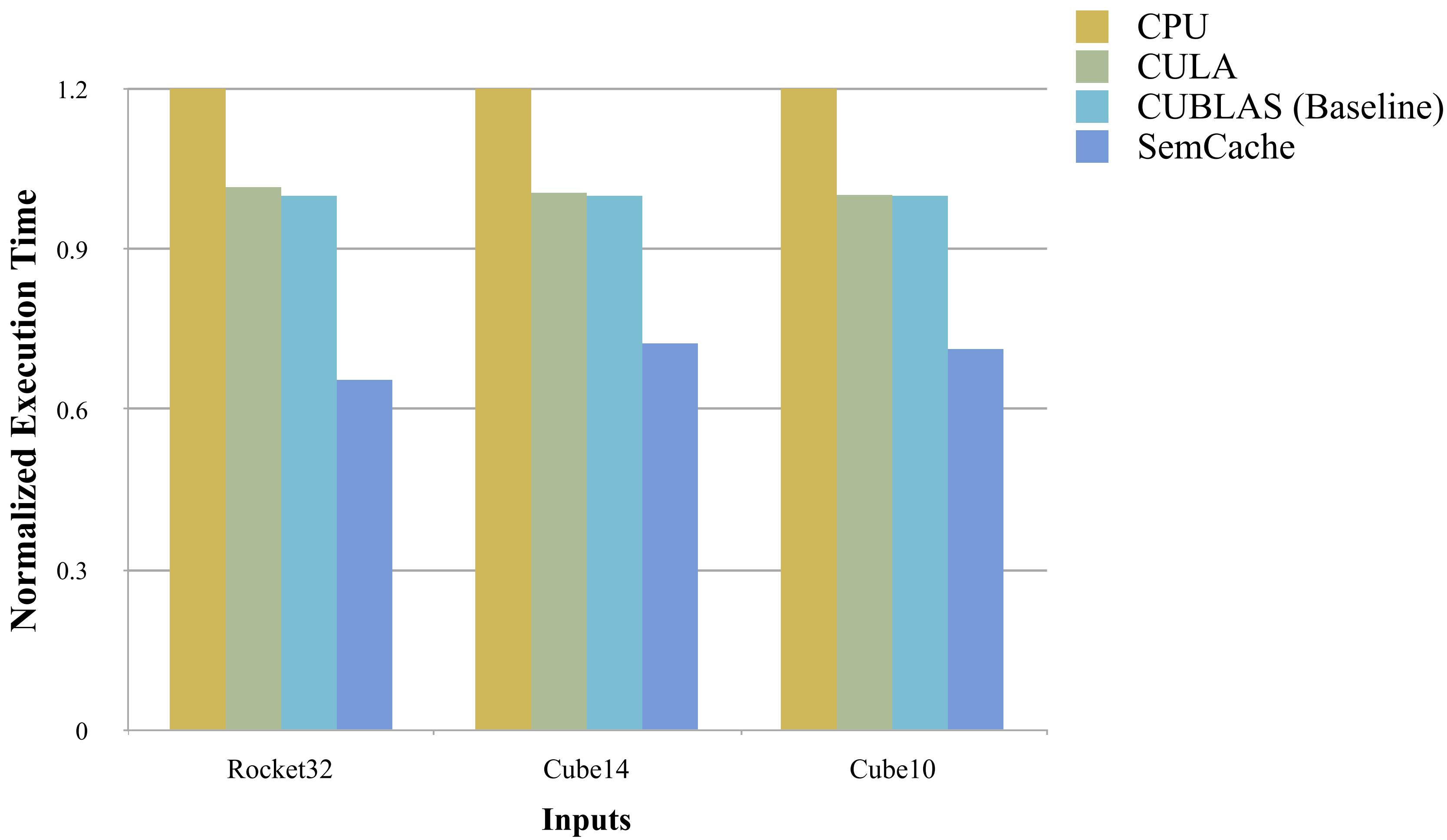
- Compiler analysis?
  - Imprecision is an issue
    - Conservative estimate of what is accessed → too much communication
  - Scalability is an issue
    - Large, modular programs; same code being used in different ways
- DSM?
  - Granularity is an issue (page based)
  - Fixed mapping between GPU and CPU address spaces
    - What if data is too big for GPU?
  - No semantic information
    - Cannot change data layout between devices

Solution: semantics-aware communication optimization

- Hybrid static/dynamic approach
- Augment libraries with information about what data needs to be read/written, any data transformations
- Semantics-aware run-time tracks data, eliminates unnecessary movement
  - Essentially, treat GPU memory as a cache
  - Tracks data *at the granularity of libraries*
  - Transparently performs data-layout changes (e.g., column-major to row-major)
  - Dynamic tracking of data means precise data movement
    - Keeps data up-to-date on both devices
    - No extra communication
- Paper presented at ICS 2013

Results

- Same computational mechanics code as before



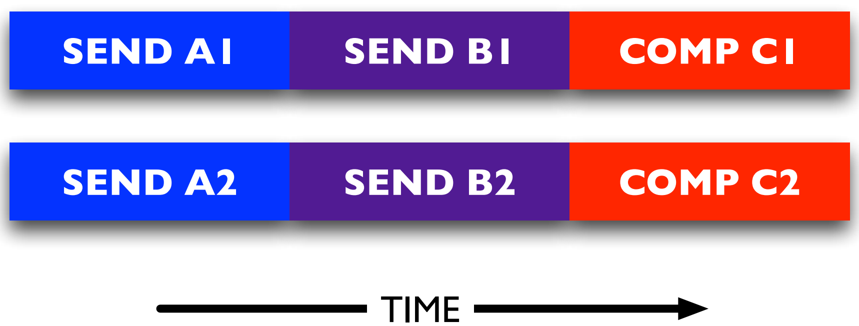
Multi-GPU SemCache

- SemCache provides automatic data management for heterogeneous nodes with a single GPU
  - Programmer writes code using regular scientific libraries that have GPU versions, SemCache manages communication between CPU and GPU
- Extended SemCache to work with multiple GPUs
- Paper under submission to Supercomputing

Challenges – Data decomposition

- Offloading to one GPU is easy: all data moves to GPU; offloading to multiple GPUs requires decomposing data and computation across GPUs
- SemCache compatible with task decompositions of library calls
  - e.g., DGEMM internally decomposed into several matrix multiplies on submatrices
- SemCache tracks *submatrices*, portions of data on each GPU, communicates submatrices as necessary

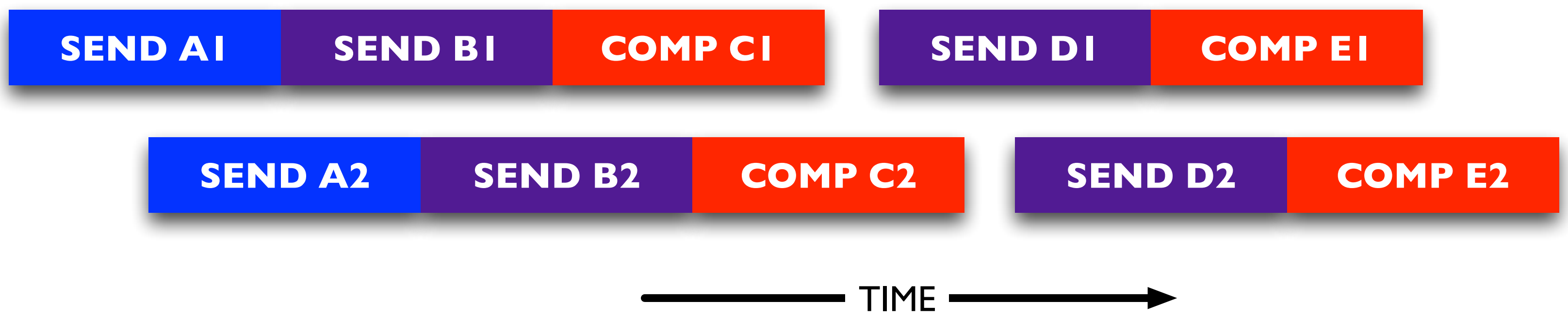
```
C = A + B
    ↓
C1 = A1 + B1
C2 = A2 + B2
```



Challenges – Synchronization

- Best performance achieved when multiple tasks run simultaneously
- Subtasks for individual library call can be synchronized easily
- Want to synchronize *across* library calls:

$$C = A + B$$
$$E = C + D$$



- Hard to do manually or at compile time because do not know what calls are coming next
- SemCache automatically inserts synchronization to make sure subtasks wait on dependences, even across library calls
- Automatically detects when data is needed on CPU, makes sure relevant tasks complete before sending data back

Challenges – Data representation

- Suppose we want to split SpMV across two GPUs

  y = A * x

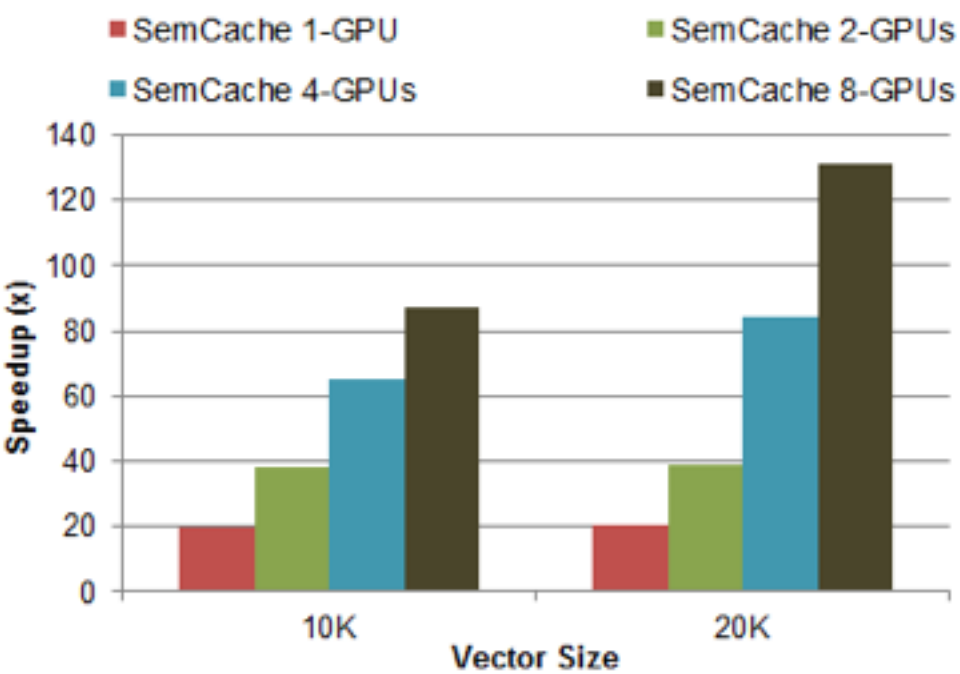
- Can decompose by splitting A by rows. Half of A sent to each GPU, all of x sent to each GPU:

  y1 = A1 * x

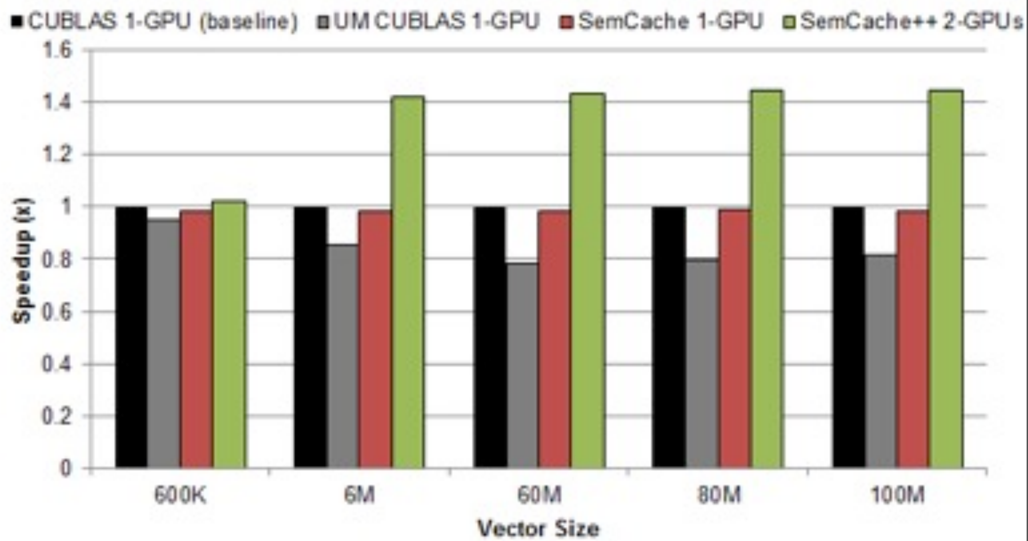
  y2 = A2 * x

- But CSR format means that A1 and A2 are not just a subset of data for A. Must recompute indexing arrays!
- SemCache's ability to make semantic links lets the decomposition of the matrix across GPUs be associated with the whole matrix on the CPU

Results



Jacobi iteration



Conjugate gradient

Use case: Kokkos + SemCache

- Kokkos is data structure library in Trilinos
- Supports transparent distribution of matrices/arrays across nodes and offloading to GPU/accelerators
- Communication currently performed manually (Kokkos directives to move data to/from GPU)
- Working to integrate SemCache with Kokkos-enabled library calls
  - Will automatically manage movement of Kokkos data structures to/from GPU
  - Will enable multi-GPU offloading (Kokkos currently supports multiple GPUs through MPI)
- First target: Kokkos-based implementation of Peridigm
- *Will provide benefits to all DOE applications written with Kokkos*

Summary/comparison

- Multi-timescale optimization techniques
  - Inspector/executor techniques have been used to schedule computations (sparse MVM, sparse Cholesky, etc.)
    - Techniques often very application specific
  - First approach to target domain decomposition problems
  - Takes advantage of semantics, but not domain specific
- Semantics-based compilation
  - Many prior approaches have targeted these kinds of optimizations, but often change representations
  - Our approach: library based program → library based program
  - "Lifting" to our representation allows more comprehensive identification of optimization opportunities
- Communication optimization for accelerator programs
  - Prior approaches have used compiler analysis, DSM-based approaches or special language constructs
  - SemCache works with any offloading library
  - Handles multiple GPUs, different data representations
  - Cleanly integrates with existing programming models (e.g., Kokkos)

SLEEC: Semantics-rich Libraries for Effective Exascale Computation