SLEEC: Semantics-rich Libraries for Effective Exascale Computation

Milind Kulkarni
Arun Prakash
Sam Midkiff

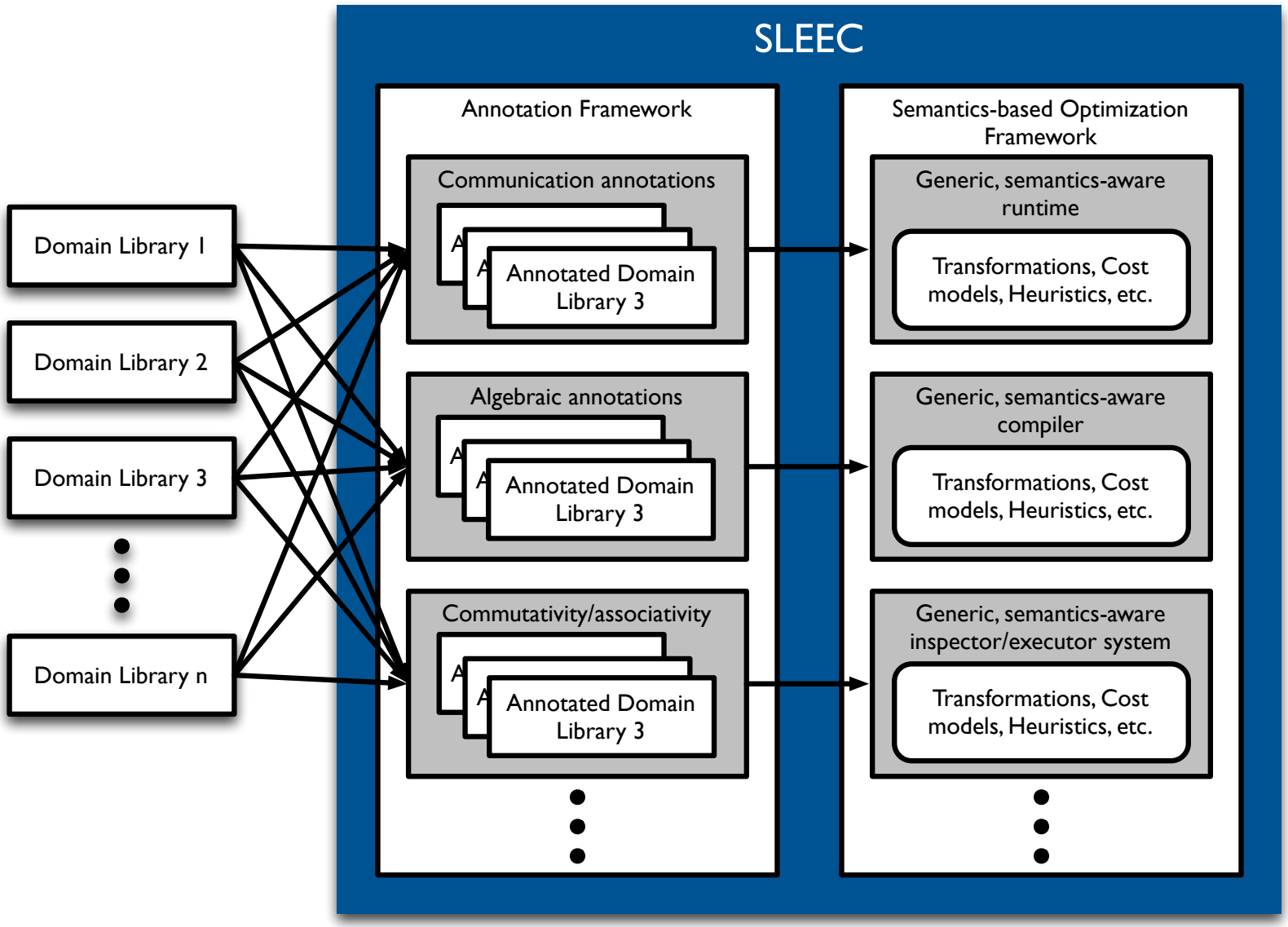
Michael Parks
Daniel Turner

PURDUE UNIVERSITY

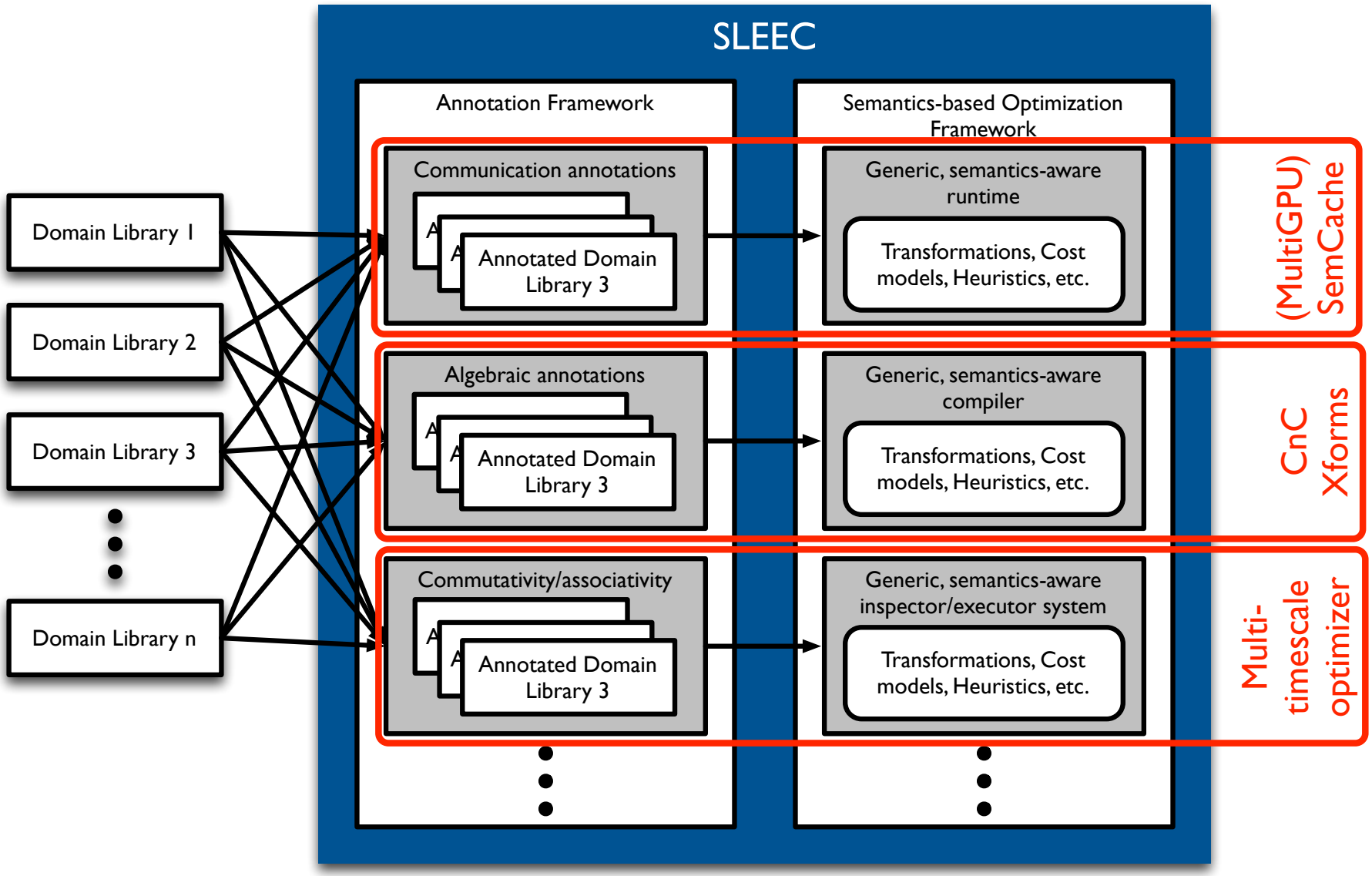
Sandia National Laboratories

https://engineering.purdue.edu/SLEEC

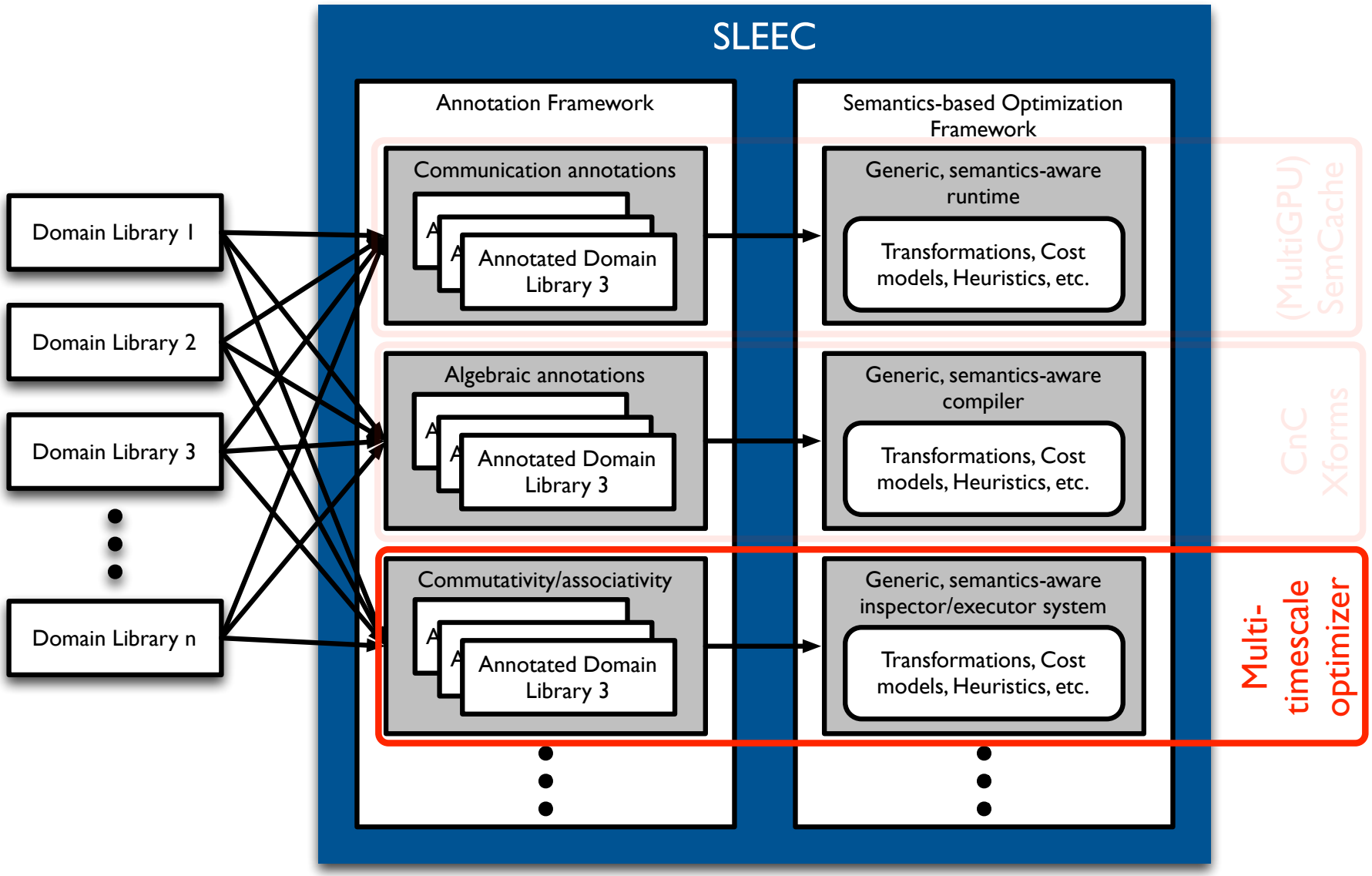
Project vision



Project status

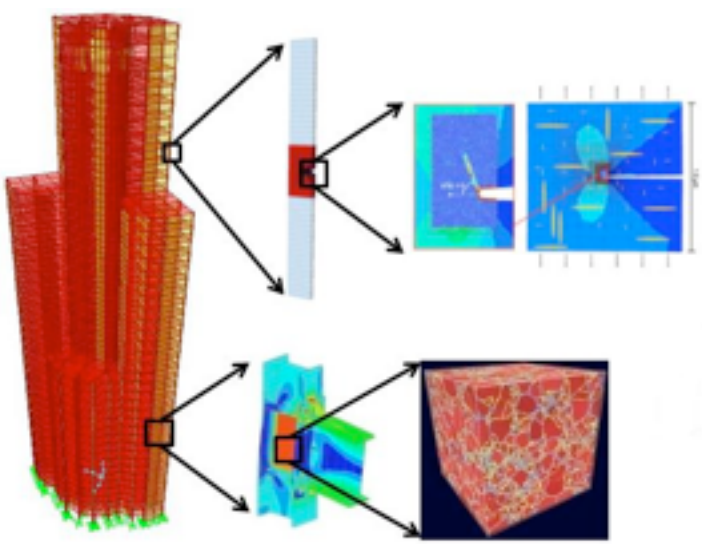


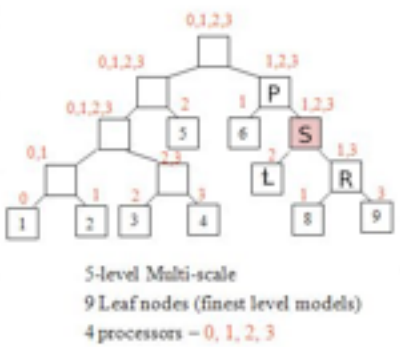
Multi-timescale optimizer

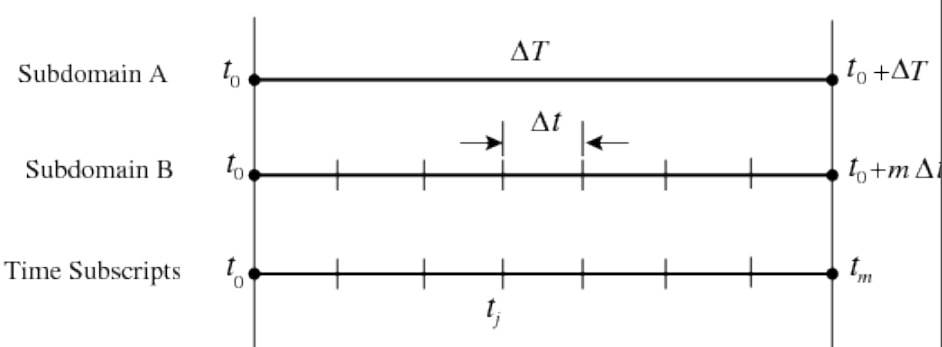


Computational mechanics

- 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)

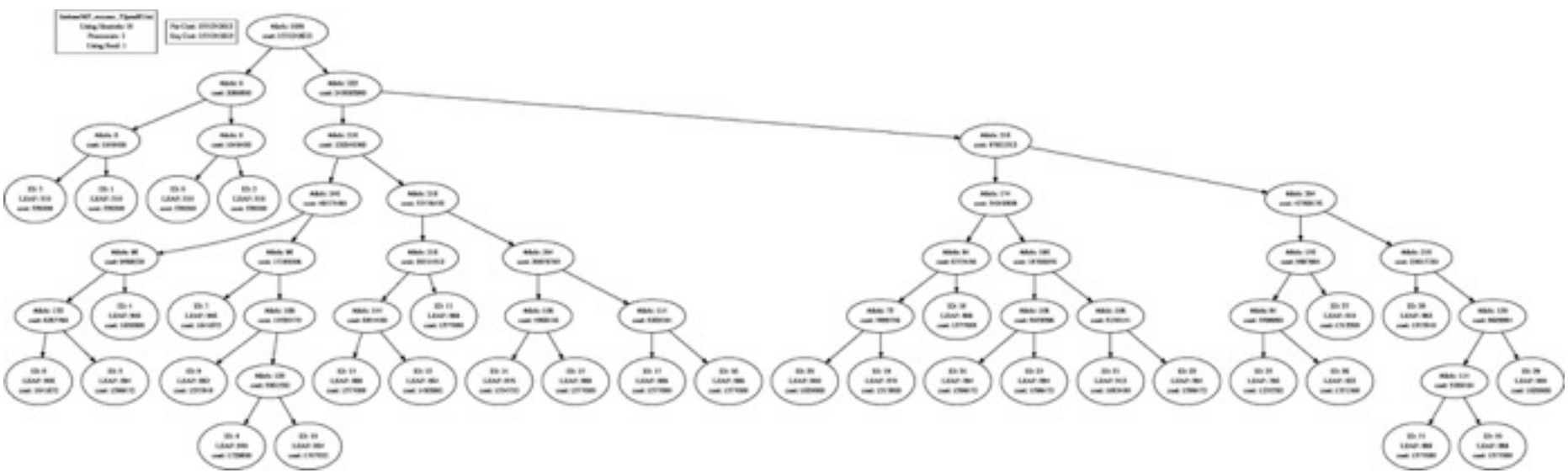






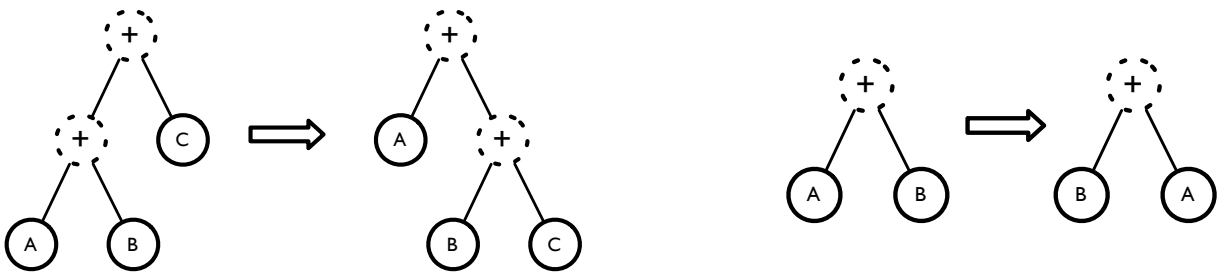
Coupling trees

- 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

Extensions

- Apply generic scheduling scheme to other domains
- Domain-aware partitioning
  - Take advantage of cost model information to decompose problem more intelligently

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 than interface in computational mechanics
  - Paper accepted to CMAME

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
- Prototype demonstrated last summer at Sandia, paper under preparation

Domain-aware partitioning

- Key problem in multi-timescale method: identifying appropriate decomposition of problem into subdomains and timescales
- Constraint: particular elements must run at sufficiently low time scale to ensure stability
  - Smaller elements (around more detailed domain features) need to run at smaller time scale
- Objectives: minimize overall computation time, maximize parallelism

Obvious solutions don't work

- Run all elements at smallest time scale, use scheduling approach we devised before
  - Large elements are simulated at too-fine granularity, wastes work
- Partition without considering time scale information
  - A subdomain has to be run at the time scale of the smallest element in it (or stability is lost) → same problem as above
- Let each element run at exactly the time scale it wants
  - Wind up with complex boundaries between subdomains at different time scales → coupling cost increases

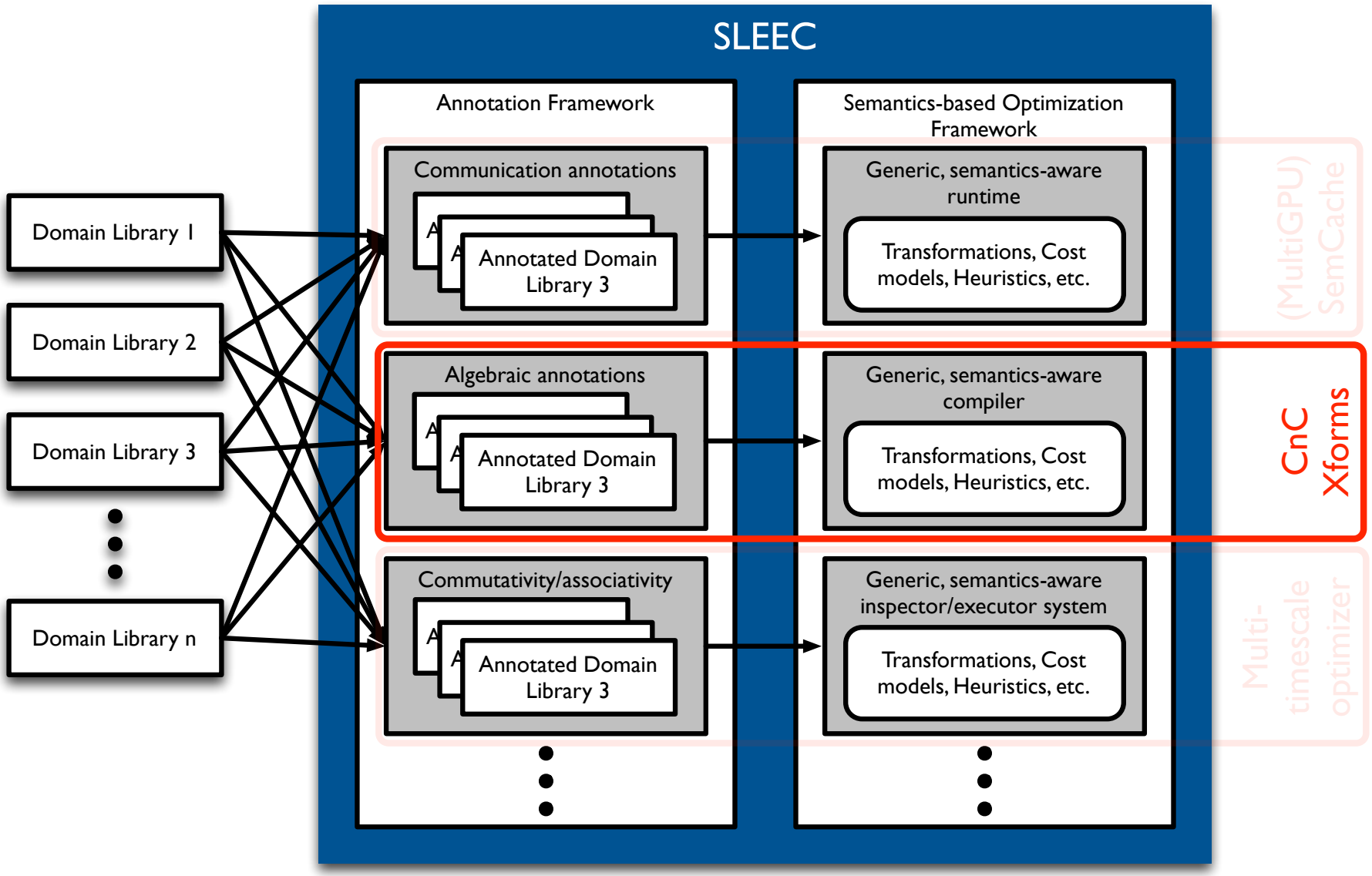
Solution: domain-aware partitioning

- Perform partitioning using modified version of traditional partitioning algorithms (multi-level partitioning a la Metis)
- Allow elements to move between time scales
  - Large elements can run at lower time scale if it reduces interface complexity enough
- Provide domain-specific cost model to partitioning algorithm
  - Partition weights based not only on number of elements in partition (as in traditional partitioners), but also based on time scale domain will run at
  - Interface weights during refinement phase based on time scale

Results

- For large problems with wide range of element sizes (e.g., beam with a notch), gives 10x performance improvement over naïve partitioning, ~20% improvement over careful, domain-aware partitioning that attempts to keep elements at the same timescale in the same subdomain
- Automatically select number of subdomains, time step ratios, etc.
- Poster presented at SC 2015, paper under preparation

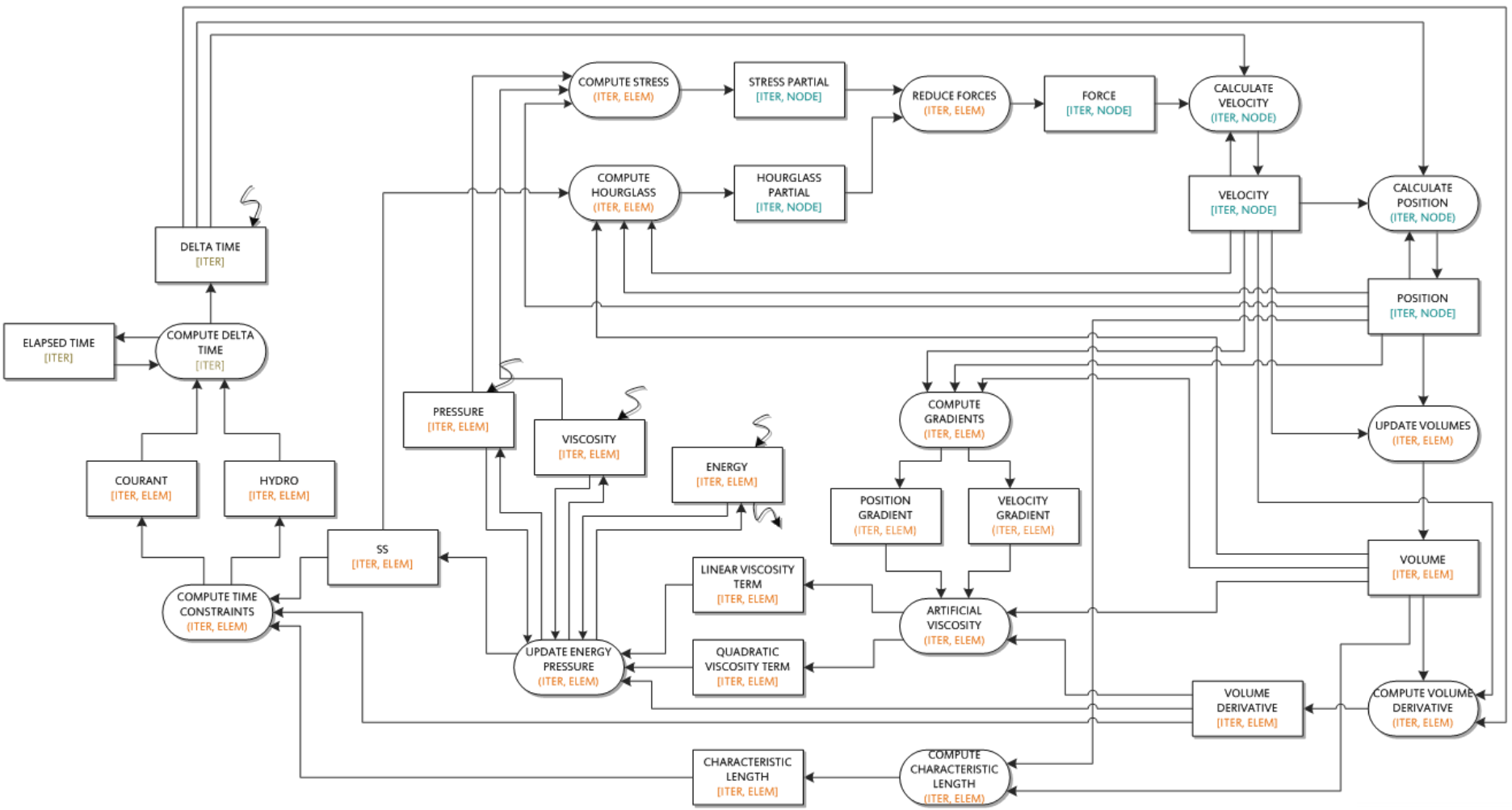
Project status



CnC Transformations

- Concurrent collections is a dataflow-esque programming model
  - Part of Traleika Glacier X-Stack project
- CnC *steps* that represent computation
  - Steps produce data and control tags consumed by other steps
  - Program representation as a *graph* of computations

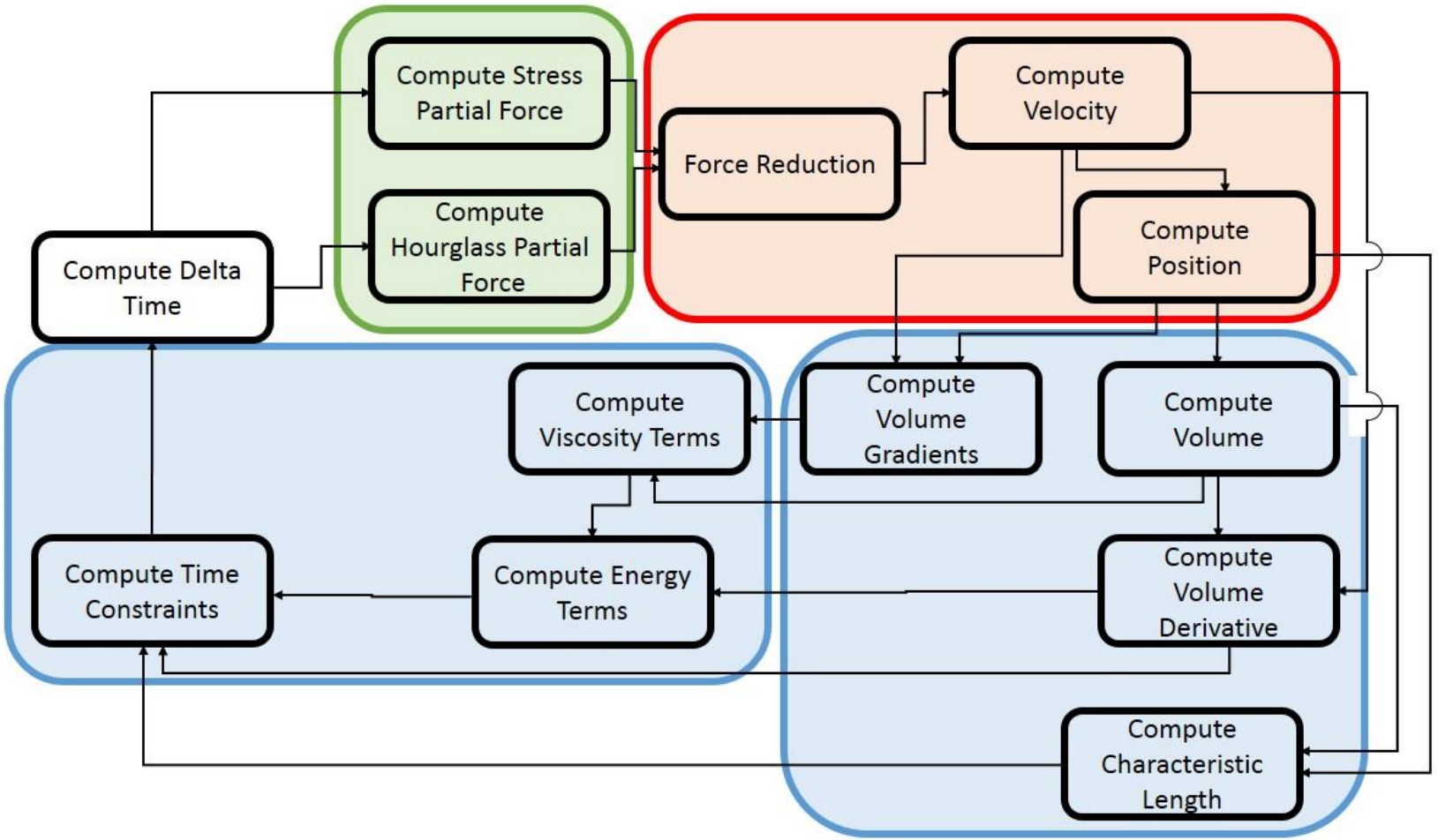
LULESH case study



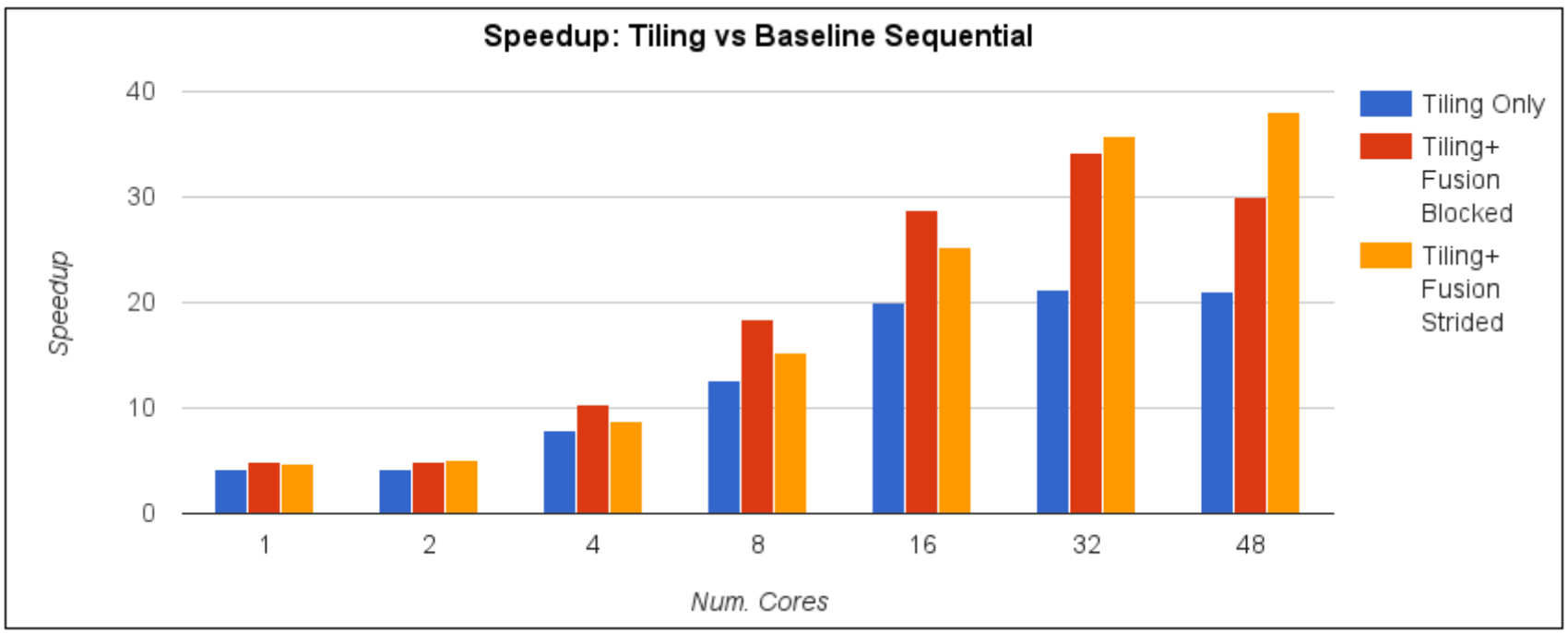
CnC Transformations

- Can consider optimizations of CnC programs as graph rewrites
  - *Step fusion*: merge together two (or more) steps across the same data item
  - *Tiling*: merge together different dynamic instances of a step across different data items
- Can use SLEEC graph-rewriting technology to perform optimizations of CnC programs

LULESH case study

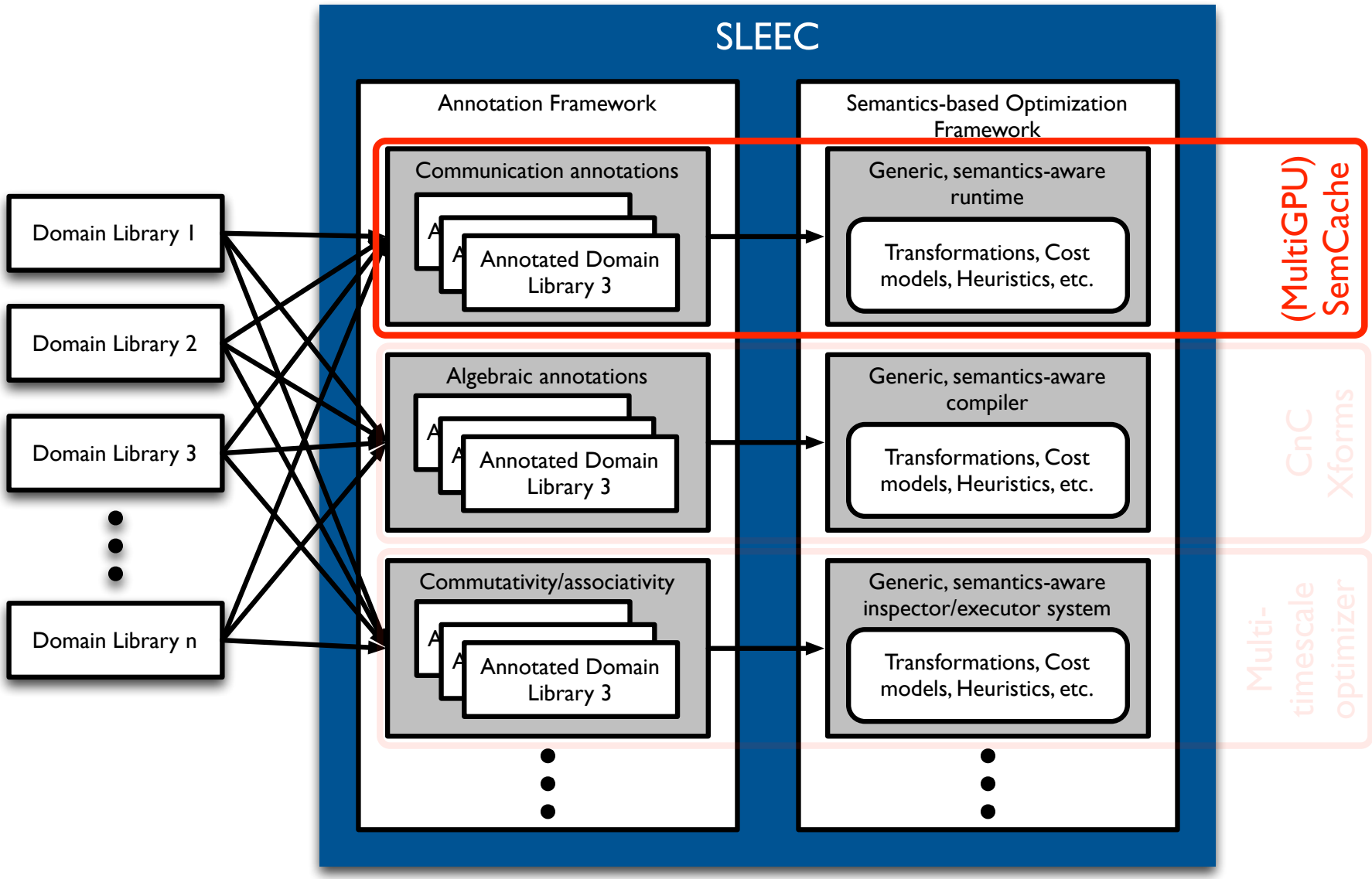


LULESH case study



Presented at WolfHPC 2015, extended version invited to IJPP

Optimizing communication/synchronization for accelerators

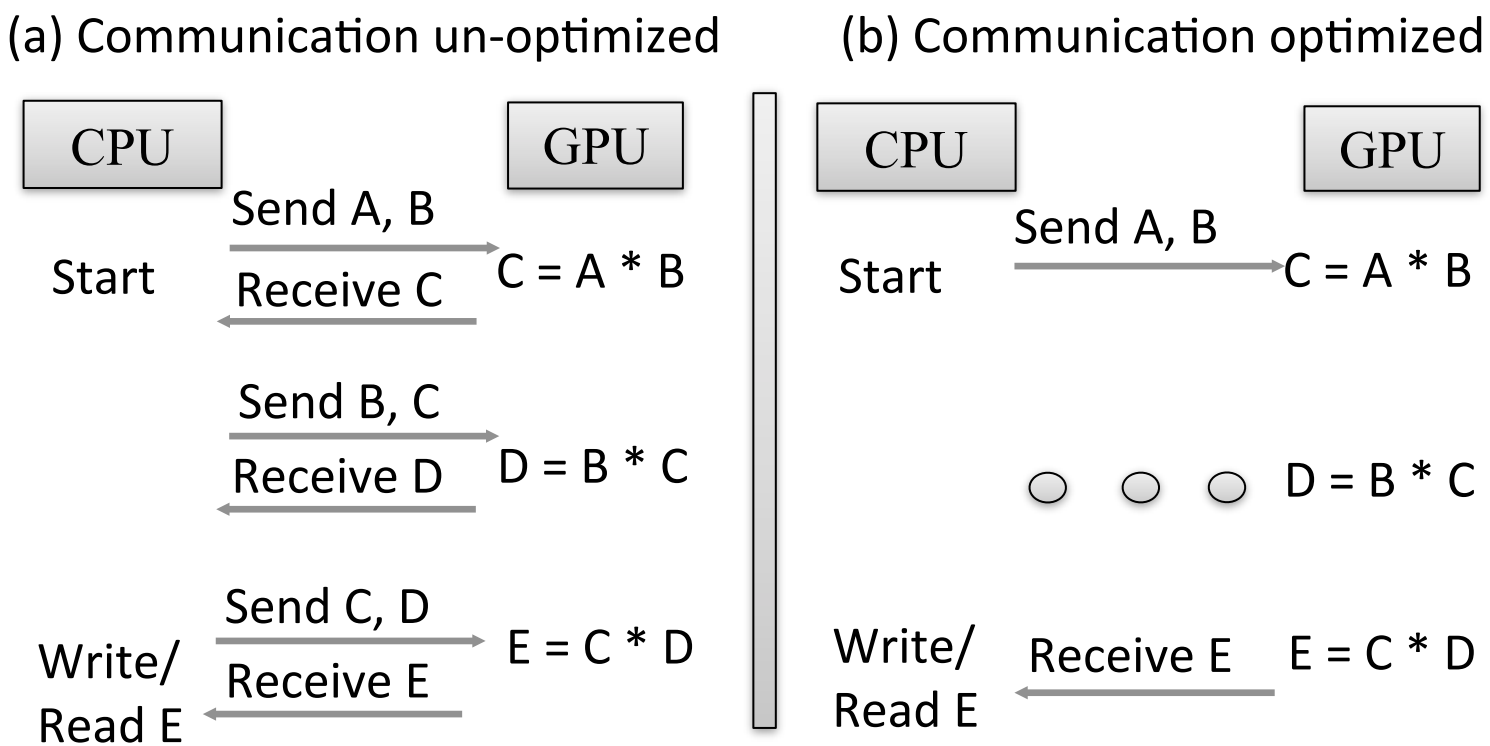


GPU offloading

- 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*



What are my options?

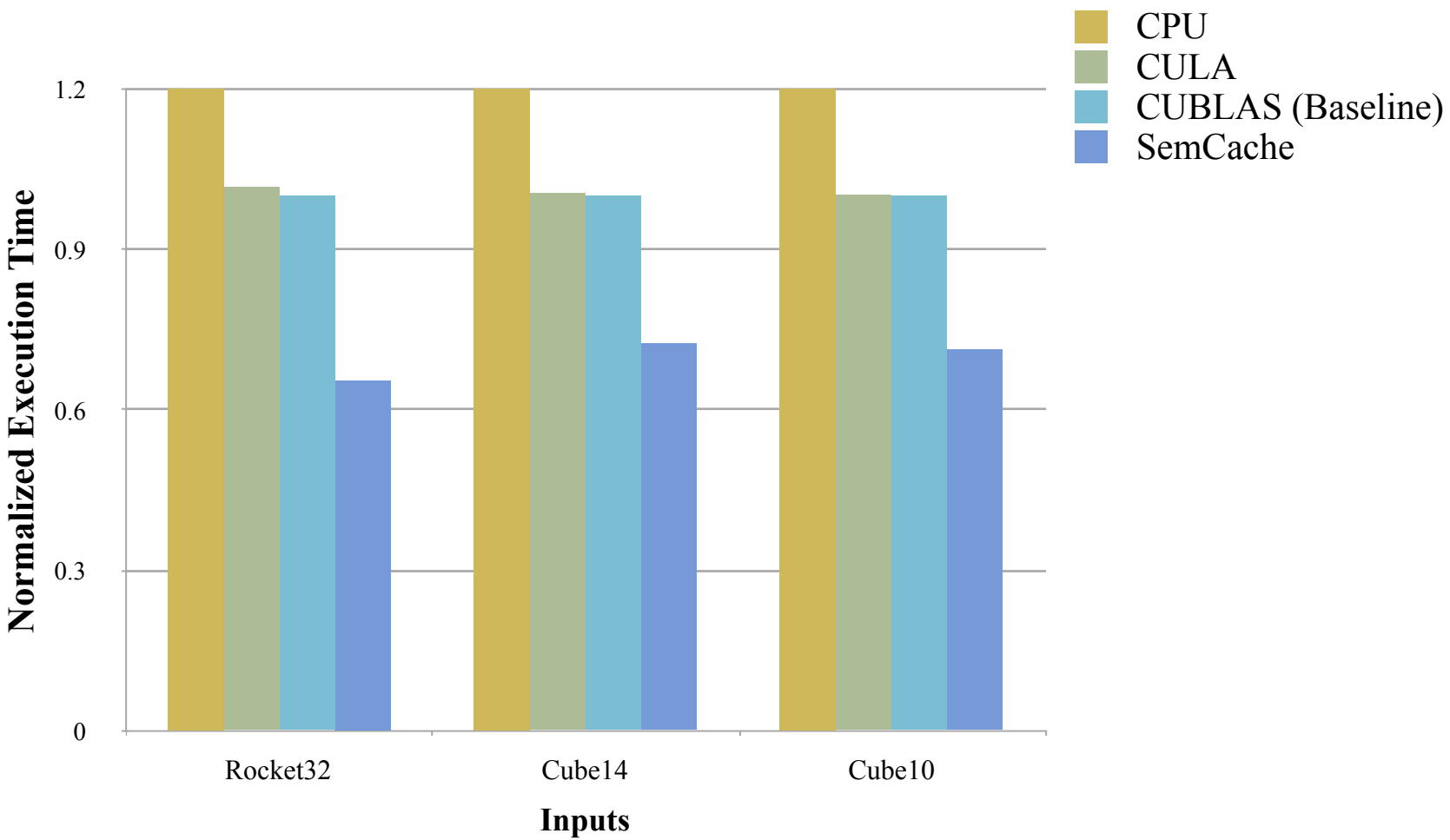
- Compiler analysis?
  - Imprecision is an issue
    - Conservative estimate of what is accessed → too much communication
  - Scalability is an issue
    - Large, modular programs; same code being used in different ways
- DSM?
  - Granularity is an issue (page based)
  - Fixed mapping between GPU and 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

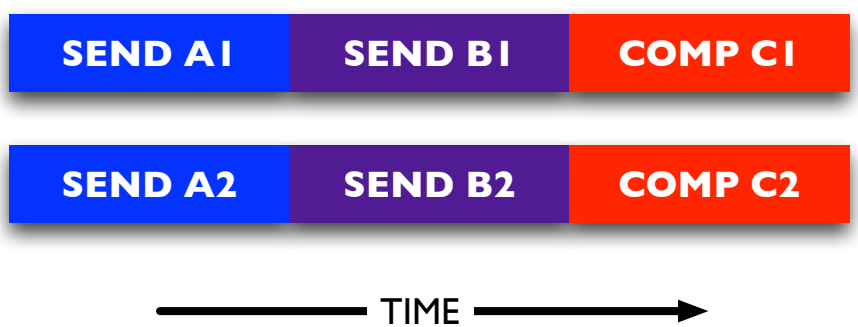
- SemCache provides automatic data management for heterogeneous nodes with a single GPU
  - Programmer writes code using regular scientific libraries that have GPU versions, SemCache manages communication between CPU and GPU
- Extended SemCache to work with multiple GPUs
- Paper presented at ICS 2015

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
```

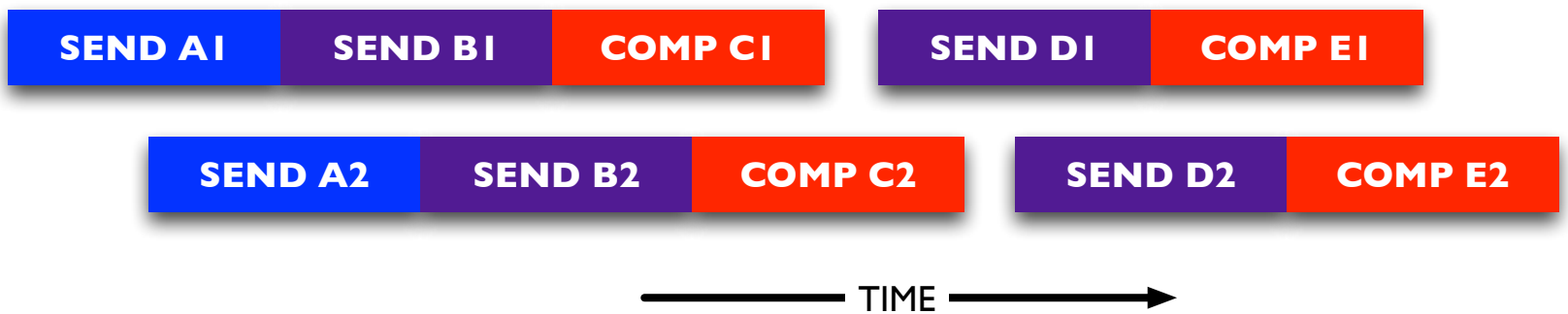
```
C1 = A1 + B1
C2 = A2 + B2
```



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:

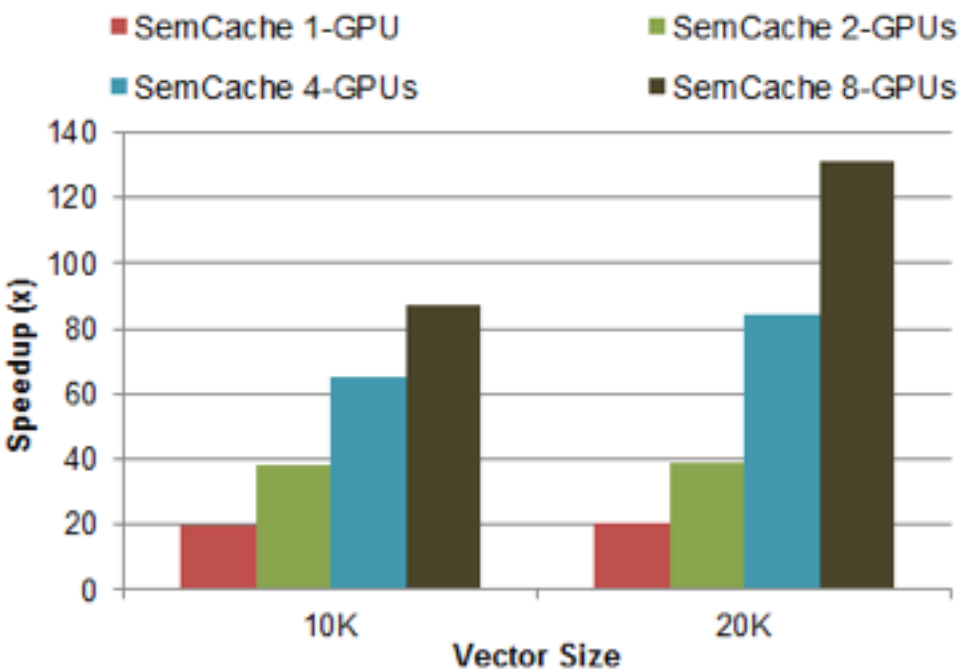
  y1 = A1 * x

  y2 = A2 * x

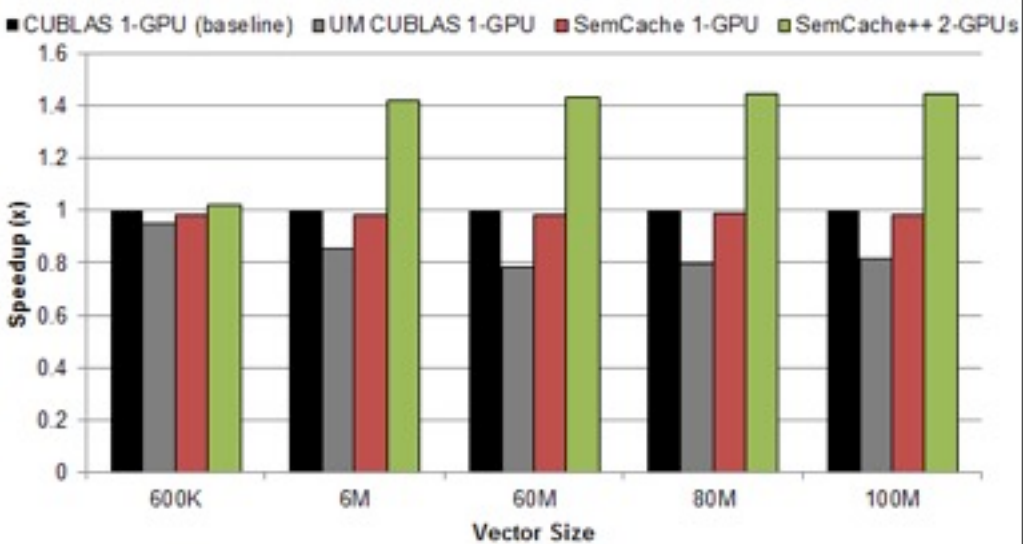
- But CSR format means that A1 and A2 are not just a subset of data for A. Must recompute indexing arrays!

- 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)

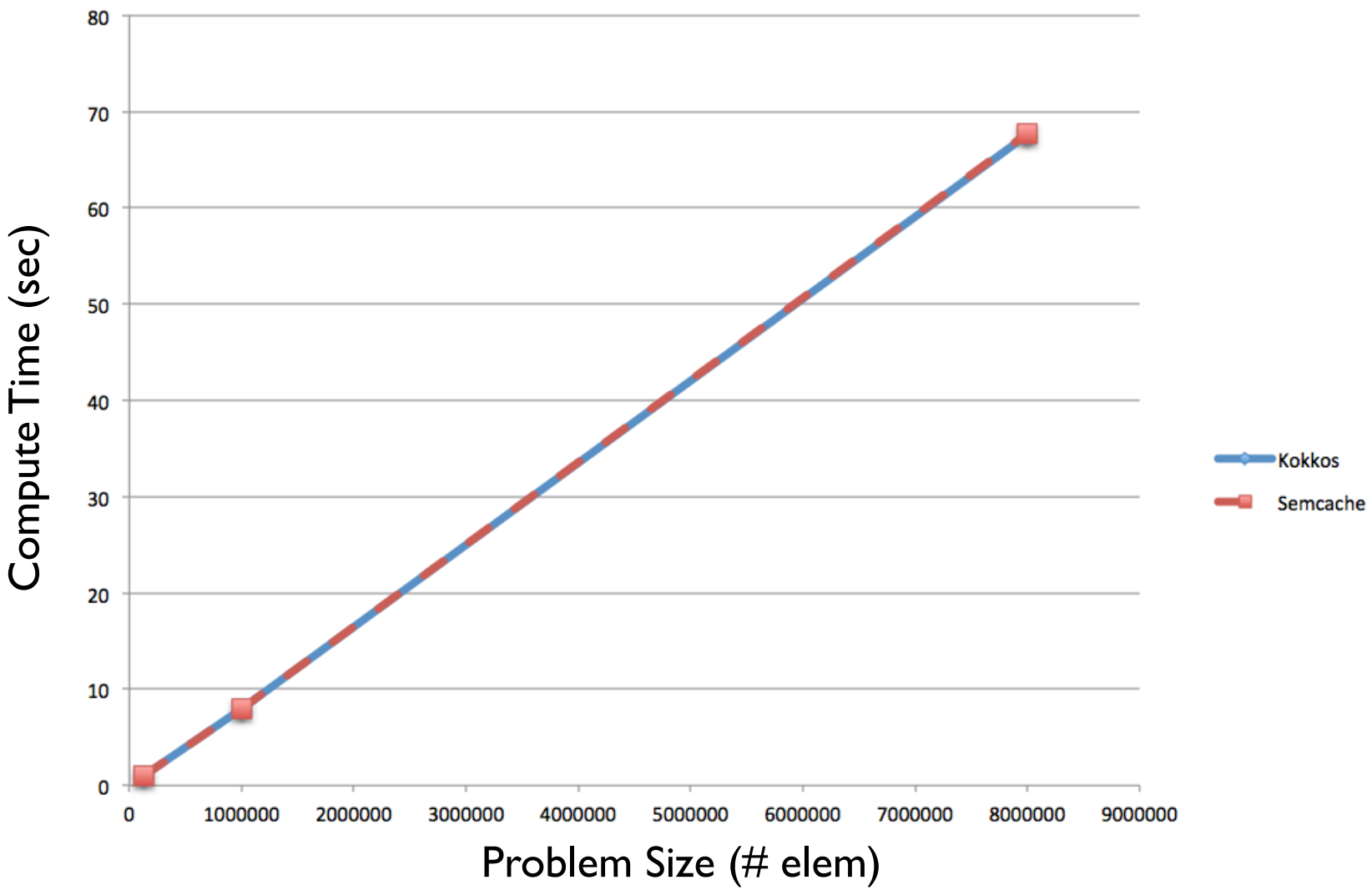
Sample code using SemCache

```
//---------------------------------------------------------
//Compute the update of a vector with the sum of two scaled vectors where:
//
// w = alpha*x + beta*y
//
// x,y - input vectors
//
// alpha,beta - scalars applied to x and y respectively
//
// w - output vector
//
template<typename VectorType>
void
  waxpby(typename VectorType::ScalarType alpha,  VectorType& x,
         typename VectorType::ScalarType beta,  VectorType& y,
         VectorType& w)
{

  int size = y.local_size<x.local_size?y.local_size:x.local_size;

  w.coefs.readGPU();
  x.coefs.readGPU();
  y.coefs.readGPU();

  if(alpha==1.0)
    Kokkos::V_Add(w.coefs.d_view,x.coefs.d_view,beta,y.coefs.d_view,size);
  else
    Kokkos::V_Add(w.coefs.d_view,alpha,x.coefs.d_view,beta,y.coefs.d_view,size);
  device_device_type::fence();

  w.coefs.writeGPU();
}
```

Simple calls to readGPU or write GPU, no need for manual testing/ synching of memory as shown here

```
  if(dev) {
    if((modified_host > 0) && (modified_host >= modified_device)) {
    Kokkos::deep_copy(d_view,h_view);
    modified_host = modified_device = 0;
    }
  } else {
    if((modified_device > 0) && (modified_device >= modified_host)) {
    Kokkos::deep_copy(h_view,d_view);
    modified_host = modified_device = 0;
    }
  }
```

SemCache vs. Kokkos on MiniFE



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
- CnC transformations
  - CnC programs look like dataflow graphs
  - Our approach: use graph rewriting techniques to implement optimizations such as fusion and tiling
- 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