SLEEC: Semantics-rich Libraries for Effective Exascale Computation

Milind Kulkarni, Arun Prakash, Vijay Pai and Sam Midkiff

Michael Parks

PURDUE UNIVERSITY

Sandia National Laboratories

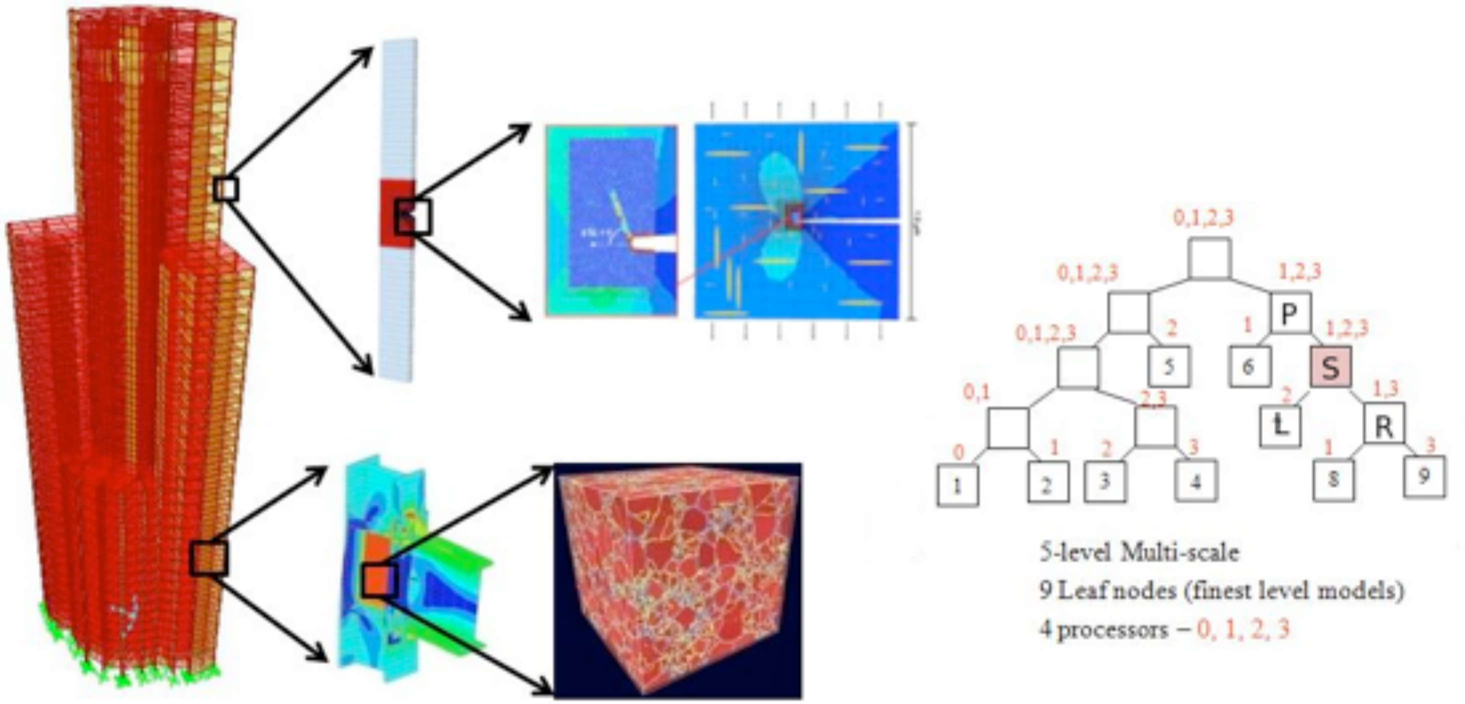
https://engineering.purdue.edu/SLEEC

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

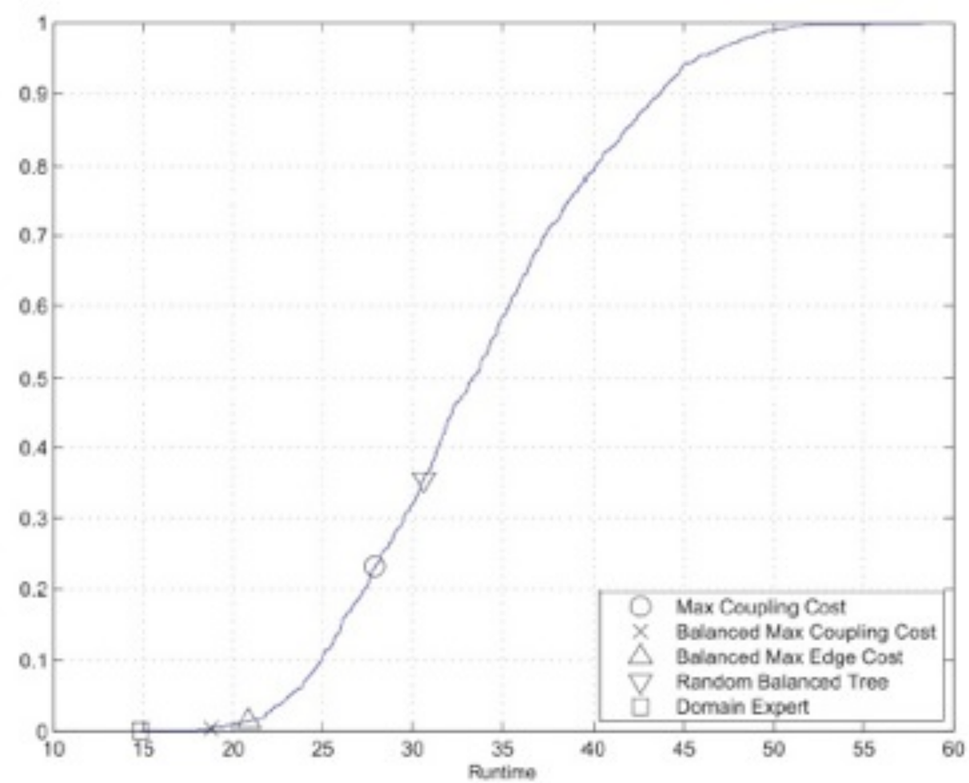
Motivation: Compositional complexity

- 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

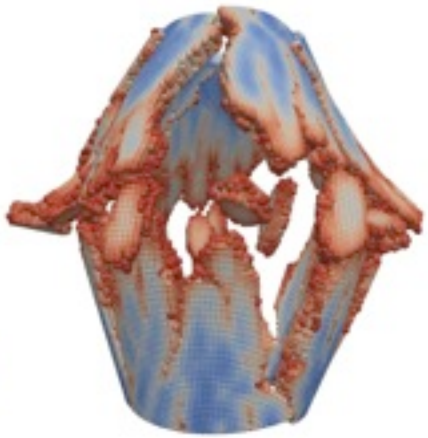
- Simple case: fixed number of subdomains, only consider how to couple them together
- Vast space of configurations: 8 subdomains → 135K possible schedules
- Large variation in performance of different orders
- Exploration of different variants requires knowledge of domain semantics, cost estimates



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

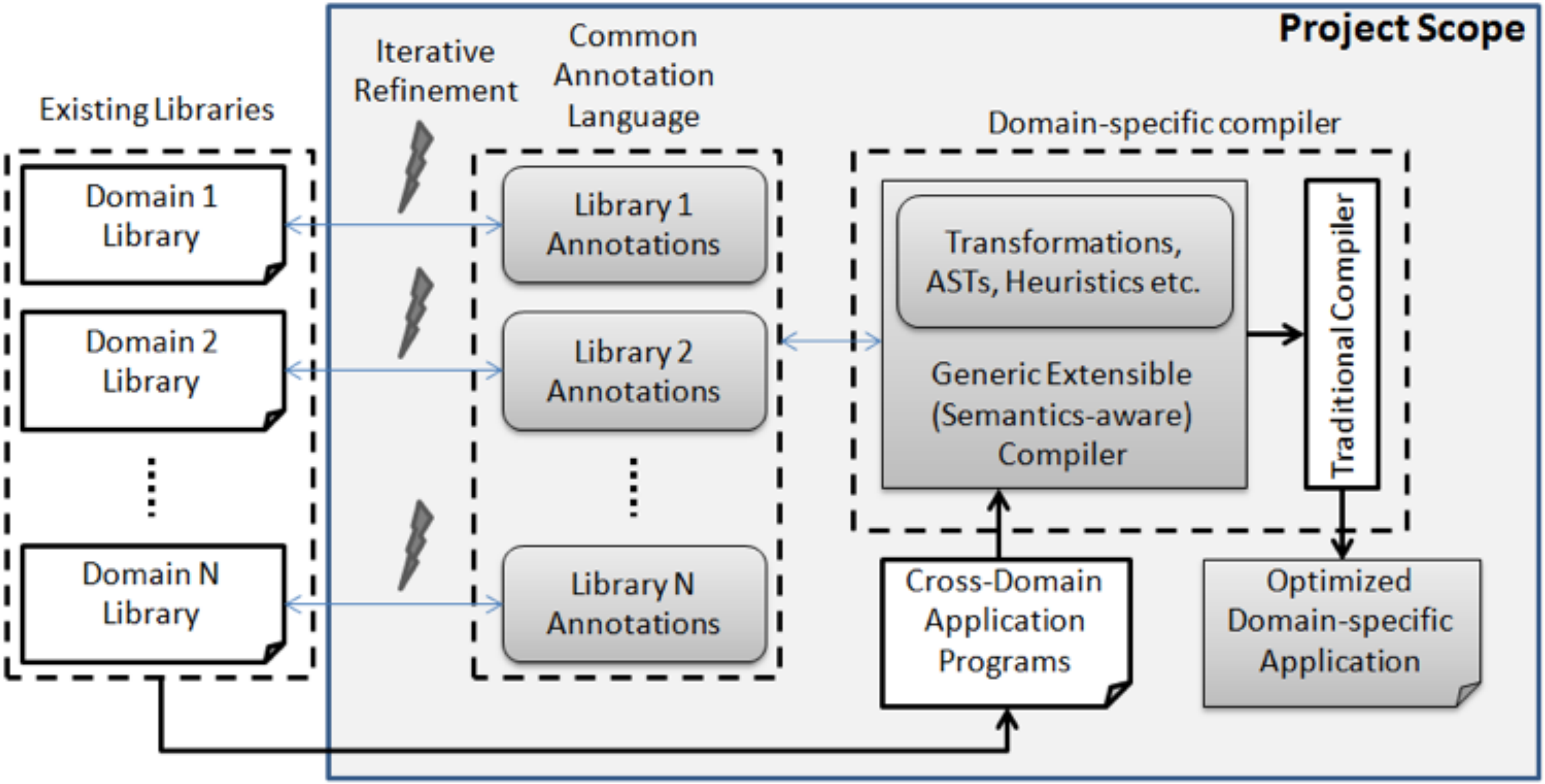


After

SLEEC: Principles

- Abstractions carried by domain libraries
  - Often a lot of semantics "in the head" of domain scientists, or even captured by library, but not communicated to compiler
  - *Need effective annotation language for capturing semantics*
- Compiler should be domain agnostic
  - Same infrastructure used for optimization and transformation regardless of domain
  - *Need common IR for capturing abstractions*
- Compiler should be able to optimize for various objectives
  - Do not want to focus solely on performance
  - *Need generic optimization ability and cost models*

SLEEC: Overview



SLEEC: Components

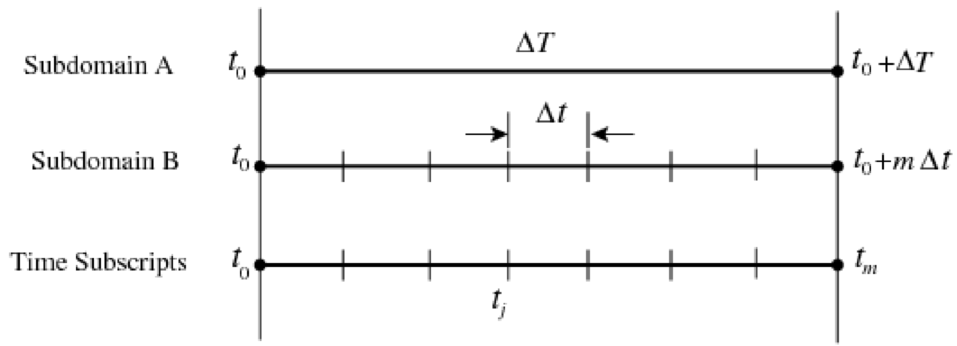
- Annotation language for capturing semantic properties of domain libraries
- High-level intermediate representation to represent programs that use annotated domain libraries
- Transformation strategies that leverage annotations to perform semantics-driven code transformations
- Optimization heuristics that use domain-specific cost models to find more efficient program variants
- Iterative refinement techniques that let the compiler work with incomplete information and infer missing information when possible

Early results/works in progress

- Optimizing computational mechanics applications
  - Taking advantage of commutativity/associativity (+ more)
- Optimizing applications with GPU offloading
  - Taking advantage of semantic equivalence between different data representations, etc.

Computational mechanics

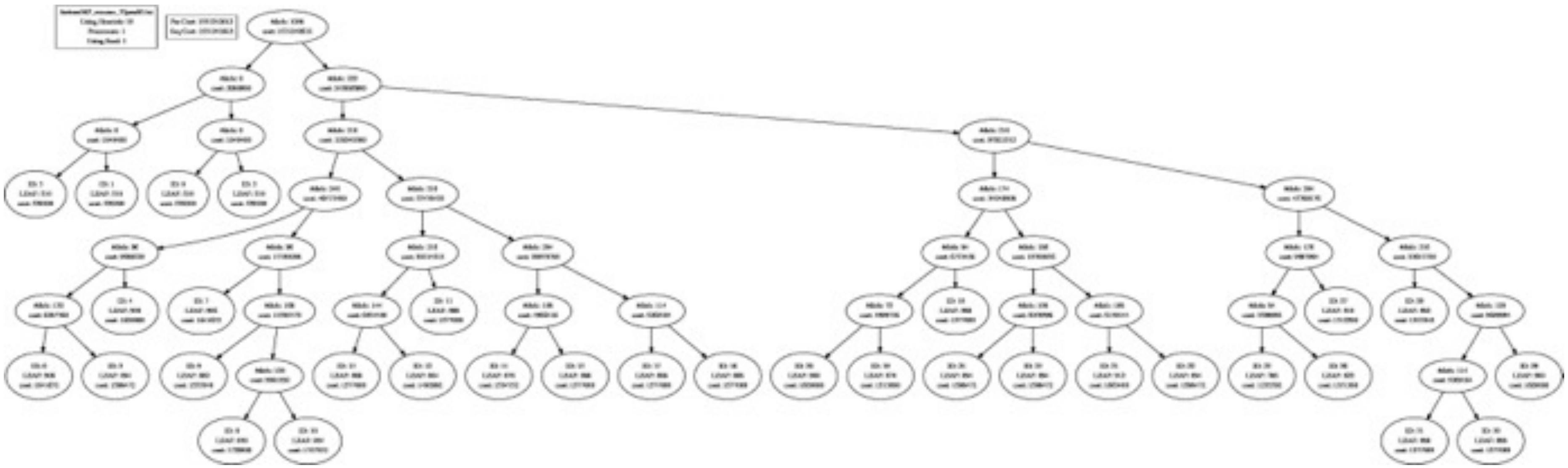
- Target: multi-scale computational mechanics codes
  - Loosely coupled problem as in intro
  - Different subdomains use different time steps (smaller time steps for subdomains that need more accuracy)



- Approach applies to other problems
  - Building multi-scale, loosely-coupled versions of peridynamics (Parks)

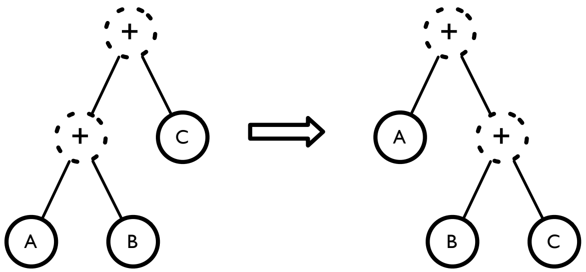
Coupling trees

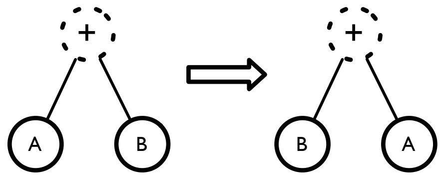
- Two basic operations:
  - `LeafSolve`: solve a single subdomain at a given time step
  - `Couple`: merge solutions from two subdomains to form "larger" subdomain



Optimizing coupling trees

- `Couple` is associative and commutative





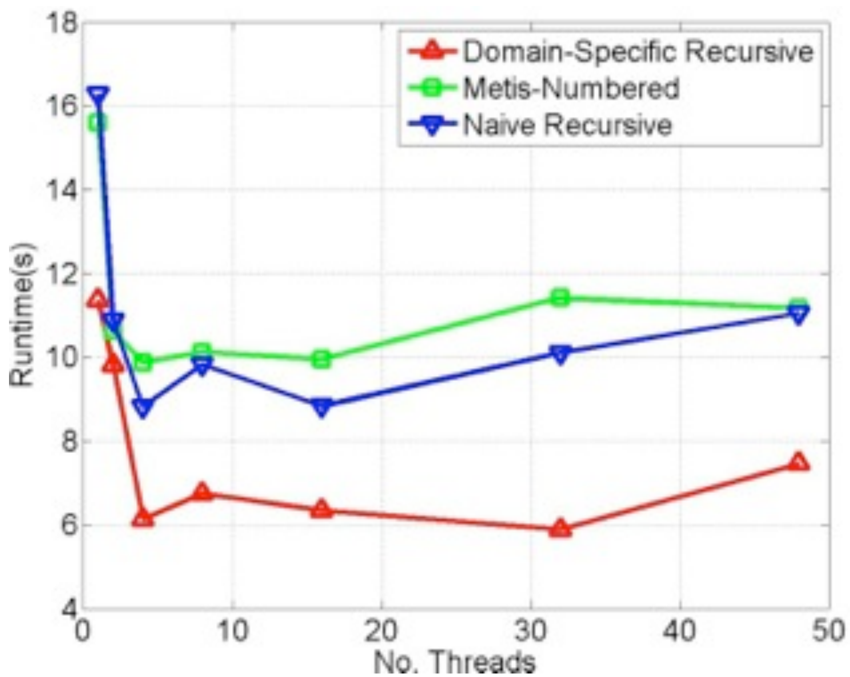
- `Couple`'s operands are also independent (parallelizable)
- Additional restriction based on domain: all domains at a given time step must be coupled before coupling with domains at other time steps
- Can be integrated into basic transformation rules:
  - Each operand has time step information
  - Time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

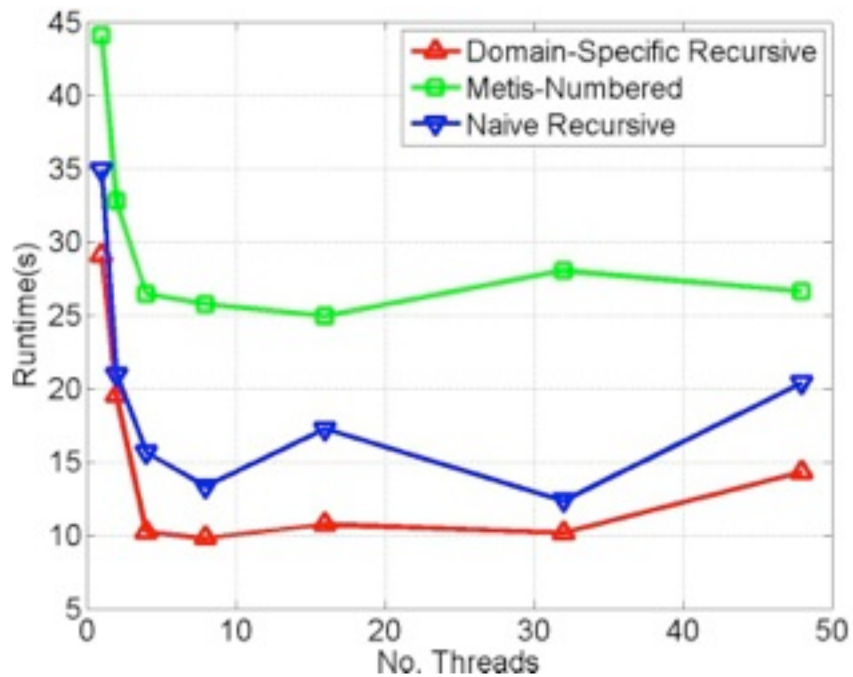
- Cost models for LeafSolve and Couple
  - LeafSolve: based on size of subdomain
  - Couple: based on size of interface between coupled subdomains, and time step ratio of subdomains
- Built heuristic based on costs
  - Attempts to produce balanced trees while minimizing overall cost and respecting constraints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

Takeaways

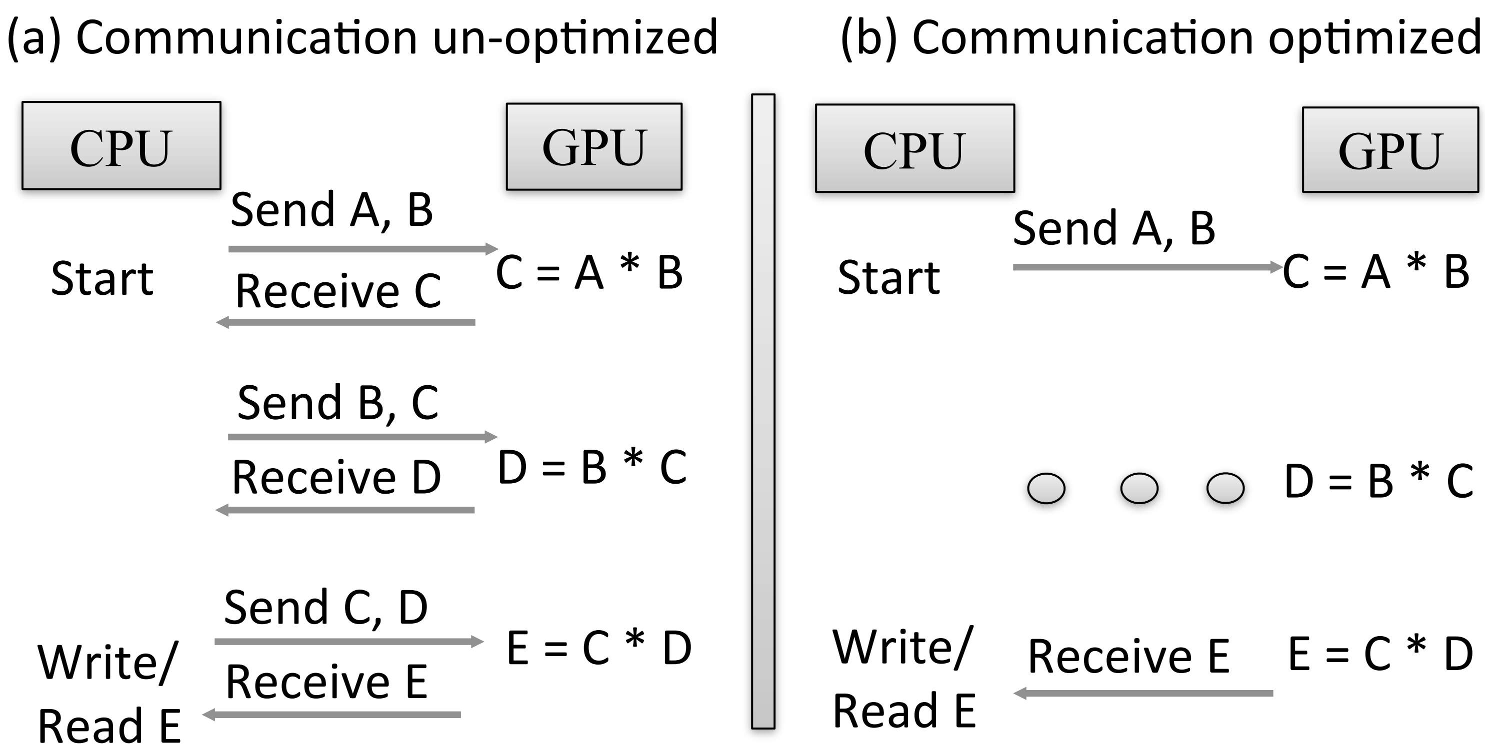
- Exploiting semantic information key to getting good performance
  - Transformation rules let system determine which coupling trees are legal
  - Cost models let system determine which orders to use
  - Both are necessary!
- Todos
  - Enrich domain semantics
    - Support changing time steps, changing decomposition in response to accuracy cost models
  - Extend to other applications

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*



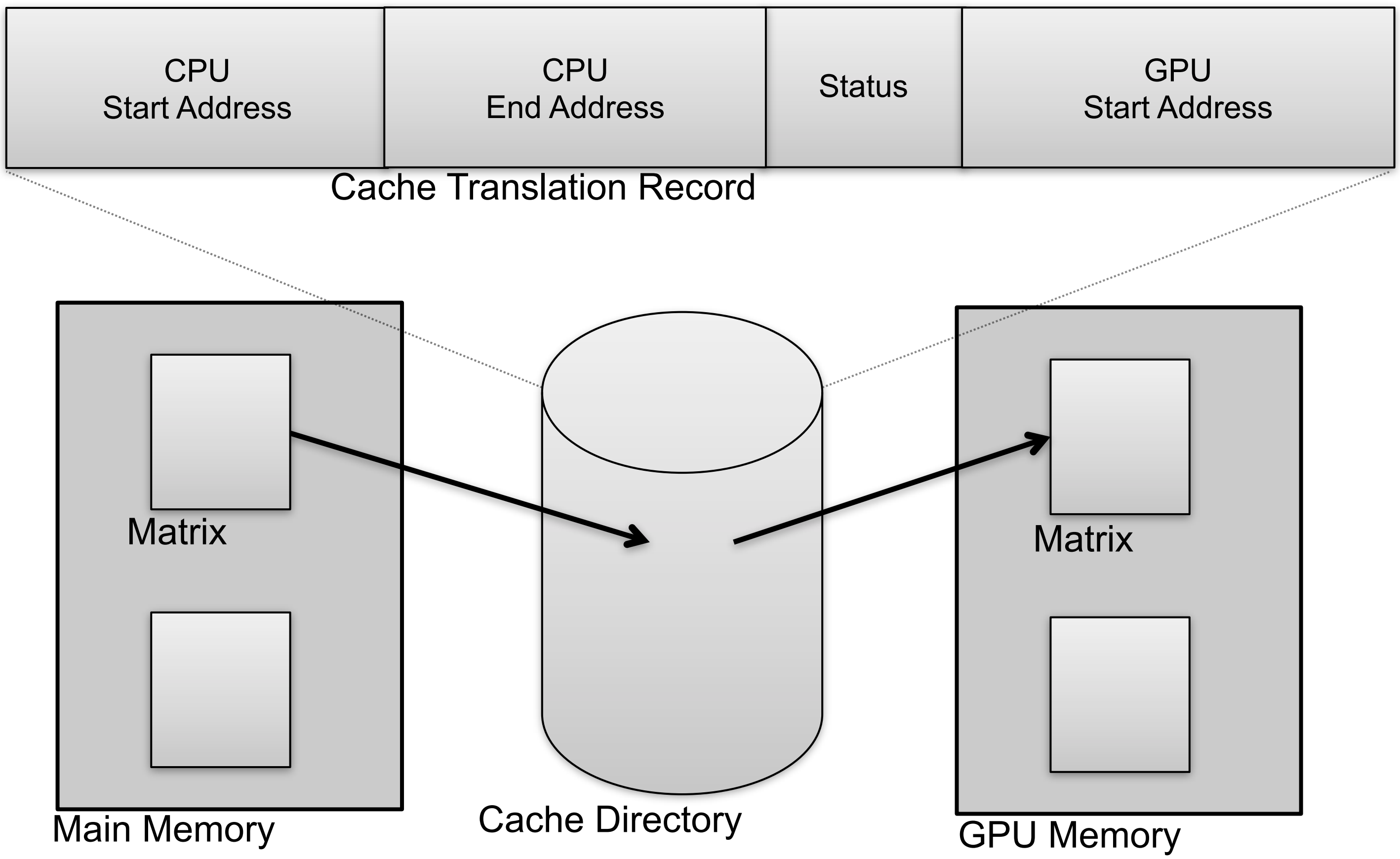
What are my options?

- 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 do more interesting mappings

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

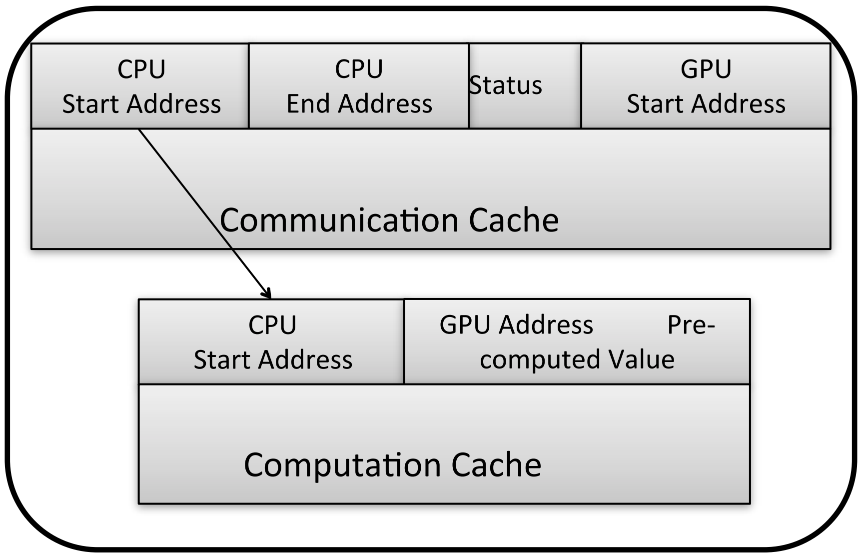
SemCache



CPU
Start Address
CPU
End Address
Status
GPU
Start Address
Cache Translation Record
Matrix
Main Memory
Cache Directory
Matrix
GPU Memory

SemCache generalized

- Does not have to be direct mapping between CPU data and GPU data
  - Can change data layout (column-major to row-major)
  - Can store pre-computed data
  - Key insight: make a *semantic* link between CPU data and GPU data



Caching Directory

Leads to drop-in library replacement

CUBLAS code to perform matrix multiply, with all communication explicitly managed by the programmer:

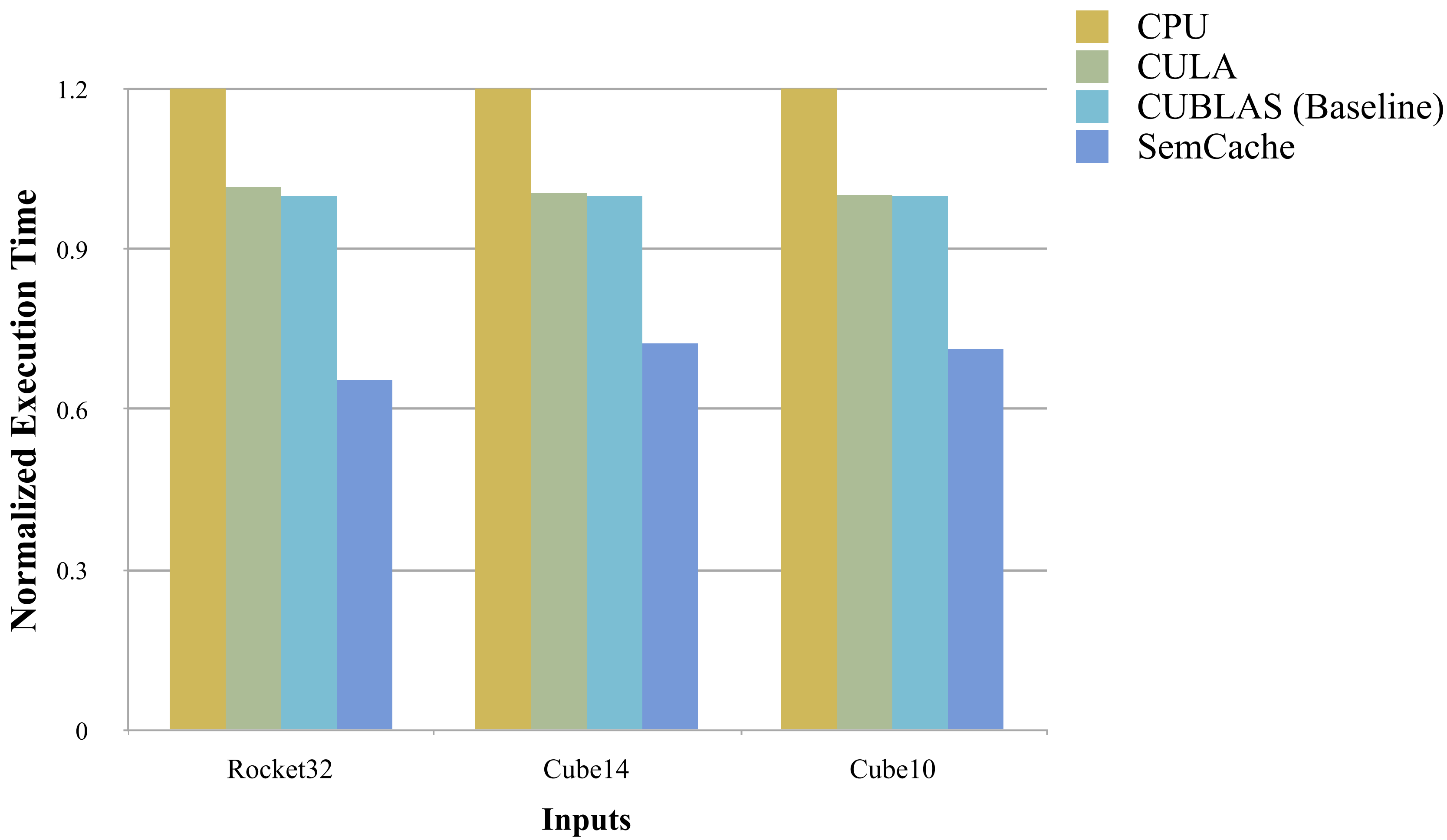
```
cudaMalloc(A) //Allocate space on device memory
cudaMalloc(B) //Allocate space on device memory
cudaMalloc(C) //Allocate space on device memory
cublasSetMatrix(A) //Move matrix A to device
cublasSetMatrix(B) //Move matrix B to device
cublasSetMatrix(C) //Move matrix C to device
cublasDgemm(TRANSA,TRANSB,M,N,K,ALPHA,
                         A,LDA,B,LDB,BETA,C,LDC)
cublasGetMatrix(C) //Get matrix C from device
```

SemCache-enhanced version of matrix multiply:

```
SemCacheDgemm(TRANSA,TRANSB,M,N,K,ALPHA,
                       A,LDA,B,LDB,BETA,C,LDC)
```

Results

- Same computational mechanics code as before



Takeaways

- SemCache is a *generic* run-time system
- Instantiated by semantic information provided by libraries
  - What data is read/written
  - Semantic link between CPU data and GPU data
- Todos:
  - Language for generating library information (currently provided programmatically)
  - Support for multi-GPU/multi-CPU systems
  - Cost-model-driven offloading decisions