SLEEC: Semantics-rich Libraries for Effective Exascale Communication

Team

Purdue University
Milind Kulkarni, Arun Prakash, Vijay Pai and Sam Midkiff
Sandia National Labs
Michael Parks

Motivation

Modern scientific applications are built out of numerous libraries, spanning a variety of domains. Finding the most efficient way to use and compose the abstractions provided by libraries to solve a problem is a difficult, tedious task. Traditional compilers do not understand library semantics and cannot optimize across abstraction boundaries.

Domain-specific approaches mitigate these issues by incorporating semantic knowledge about domains into languages, compilers and runtimes. This allows properties of domain abstractions to be leveraged when optimizing an application. However, most domain-specific approaches are *too specific*: each domain has its own languages and compilers, its own representations of abstractions and its own optimization strategies. This makes it difficult to target applications that span multiple domains, including emerging computational science applications such multi-scale, multi-physics simulations.

What is needed are approaches that allow *generic* compilers and runtime systems to incorporate domain knowledge without being specialized to a single domain.

Goals and Principles

We are building a ***generic, extensible compiler infrastructure*** that incorporates semantic information from domain-specific libraries to enable transformations that leverage domain-specific properties of library methods. Rather than building domain-specific compilers for each domain, our extensible compiler *becomes* a domain specific compiler for a domain when paired with domain-specific libraries.

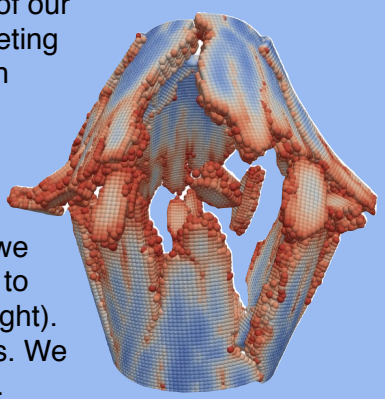
The SLEEC vision is based around three key principles:

- ***Domain-specific information should be conveyed by libraries.*** Domain semantics are best-understood by the scientists writing the domain libraries, so any semantic information should be provided as part of a library's specification or interface.
- ***Compilers should be domain agnostic.*** Rather than building *ad hoc* compiler technology for each domain, the compiler should transform programs in that domain without having been explicitly specialized for that domain.
- ***Compilers should optimize for multiple objectives.*** A compiler should be parameterized by the optimization target (performance, energy-efficiency, communication-minimization, etc.).

Unlike many domain-specific approaches, one of our primary goals is to ensure that our compiler can ***optimize across domains.*** This requires that the compiler understand transformations and optimizations in a domain-oblivious manner

Challenge applications

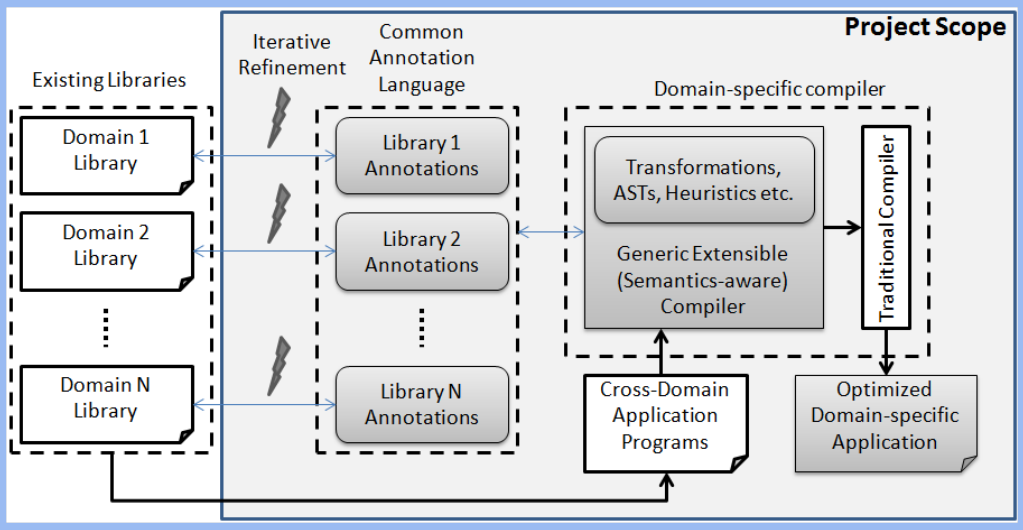
To demonstrate the utility of our infrastructure, we are targeting two specific application classes, both of which feature complex domains with rich semantics. First, we are examining loosely-coupled, multi-scale computational mechanics, which involves multiple temporal- and spatial-scales and multiple solution and coupling strategies, the various combinations of which lead to hundreds of thousands of possible solutions. We will use the semantics of domain decomposition, solving and coupling to find efficient solutions. Second, we are studying the peridynamics code Peridigm, which uses peridynamic methods to simulate large-scale physical systems (such as the fragmenting cylinder to the right). This application is built on the Trilinos components, a highly-tuned set of libraries. We will use the semantic and locality properties of these libraries in our optimization.



Project overview and approach

We will develop a unified intermediate representation (IR) that can capture high level abstractions provided by domain libraries. The IR will represent programs built with domain libraries as expression trees that capture the dataflow of a program, with library methods as operations in those trees.

Domain libraries will provide translations to and from our common IR; annotations that describe high level properties of operations that drive transformations (e.g., commutativity); and cost models that capture various cost metrics for library methods.



Our generic compiler can then use the annotations to transform a source program, generating a space of equivalent programs. The compiler then uses the cost models to search the space of equivalent programs, finding a low-cost implementation according to a user-provided objective (e.g., to minimize communication costs).

Because all libraries use a common annotation language, a program using multiple distinct libraries can be described in the unified representation of our IR. Hence, our compiler can apply transformations that span multiple libraries or even domains.

Impact

SLEEC is expected to impact exascale programming along a number of dimensions:

- ***Programmability.*** SLEEC allows programmers to use high level domain libraries while still allowing compiler-driven optimization, letting programmers focus on correctness instead of performance.
- ***Performance portability:*** SLEEC can enable library selection, or replacing certain domain constructs with variants that perform better on different architectures.
- ***Scalability.*** SLEEC's transformations allow code to be transformed to take advantage of accelerators, or to enhance locality and parallelism.
- ***Energy efficiency.*** SLEEC's cost models can capture energy efficiency, letting the compiler's parameterized optimization strategy search for energy-efficient, rather than high-performance, implementations
- ***Resilience.*** SLEEC's cost models can capture fault-tolerance properties, letting the compiler automatically select more resilient library implementations only when necessary.

Research challenges

- *How should the annotation language look?* It should be rich enough to capture important domain properties, but simple enough for domain experts to use.
- *How should the compiler perform its optimization?* We require heuristics and search strategies to quickly prune a large search space of program variants and find a low-cost implementation.
- *How can we deal with multi-domain programs?* We must be able to perform optimizations across domains. How do the IR and annotation languages need to be modified to accommodate cross-domain transformations?
- *How can we deal with incomplete information?* The infrastructure must gracefully degrade even if only incomplete annotations are provided. Can we infer missing annotations?

http://engineering.purdue.edu/~milind/sleec

PURDUE UNIVERSITY

Sandia National Laboratories

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.