Wrapper Wrap Up

Todd Gamblin

Drew Bernat

Bronis de Supinski

Madhavi Krishnan

Dave Montoya

Bernd Mohr

Dan Quinlan

Martin Schulz

Bill Williams

CScADS Workshop on Tools 2011

Major Issues with Wrapping

- **Semantic details about wrapped libraries**
  - How to specify semantics in a metadata file?
  - How to standardize this data?
- **Template languages to describe wrapping**
  - Basic syntax (personal preference)
  - How to develop language constructs to leverage semantics
- **Mechanisms for function interception**
  - Need a common interface in generated code
  - Should support multiple backend interception methods
    - Binary rewriting
    - dlsym, etc.
    - Generating name shifted interfaces automatically

Expressing Library Semantics

- **Existing ways to do this:**
  - Scalasca
    - XML file that gives information on parameters for MPI
  - LLNL wrap.py
    - Some a priori semantic knowledge
    - Nothing standard yet

- **Proposals:**
  - **Come up with some unified format for library function semantics**
    - First, support MPI semantics from existing formats.
      - In/Out parameters
      - Collectives
      - Parameter Types
  - **Extend format to specify same types of information for other libraries**
  - **Think about template language constructs to leverage these.**

Template Languages

- **LLNL, Scalasca languages not so different at this level**
  - Both only support MPI
  - LLNL uses braces, expression language
  - Scalasca uses C + pragmas
  - Leveraging of semantics is limited

- **Not clear there is a need to unify these**
  - Languages are essentially macro/template languages
    - Still evolving advanced features
    - Standardization at this level doesn't make much sense yet.
  - Let domain-specific languages, etc. evolve then reconsider

- **Talk again about unification once we have:**
  - Advanced language features to leverage semantics
  - Ways to wrap generic APIs

Mechanisms for function interception

- **Too many existing mechanisms for doing interception**
  - Name shifting
  - dlsym, other linker support
  - Generating code
  - Rewriting static binaries

- **Each of these is better in different environments:**
  - Static binaries
  - Dynamic binaries
  - Different compilers and link environments

- **Proposal: Generic interface in generated code, dyninst handles mechanism**
  1. **Generated code uses simple name shift interface**
     - Come up with a name shifting convention for calls
  2. **Dyninst backend converts this to convention for the particular environment**
     - Choose mechanism based on host machine