Parallel Object Contracts for High Performance Computing

Kosta Damevski\(^1\), Tamara Dahlgren\(^2\)

\(^1\)Virginia State University
\(^2\)Lawrence Livermore National Laboratory
Dealing with Software Complexity

- Scientific and HPC codes tend to last for a long time (decades) and often need to evolve
- Composition is key to improving programmer productivity
  - Building larger software from smaller parts (objects, components)
  - Opportunity to reuse highly tuned libraries
- As software size grows, integration errors become prevalent and difficult to debug
  - Design-by-Contract is a tool that can help trap errors at the unit boundary
Benefits of Design by Contract

- Executable component contracts have several advantages:
  - Improve application quality, by promoting safer reuse
  - Provide a better description of a component’s functionality at the interface level
  - Commonly consist of method preconditions, postconditions and class invariants

```
method invocation

? precondition + invariant

results

postcondition + ?

exec method
```
Our Work

• Propose a set of new contract clauses and operations for parallel objects / components
  • Build on the existing design-by-contract facility in the Babel compiler
  • Most existing efforts deal with concurrency, but not parallelism
  • Still in the prototype phase
    • We are still actively looking for ways to enhance and improve
  • Strive for low-overhead in order to maximize acceptance by users
Talk Outline

- Introduction
- Background
  - Parallel Objects/Components
  - Design by Contract (with Babel)
- Proposed Parallel Contract Clauses
- Preliminary Evaluation
Talk Outline

• Introduction

• Background
  • Parallel Objects/Components
  • Design by Contract (with Babel)

• Proposed Parallel Contract Clauses

• Preliminary Evaluation
Background: CCA, SIDL, Babel

- **CCA (Common Component Architecture)**
  - component model tuned for HPC
  - low overhead support for parallelism, complex numbers, multidimensional arrays
- **SIDL (Scientific Interface Definition Language)**
  - language to define CCA component interfaces
  - expresses classes, interfaces, methods, etc.
- **Babel**
  - compiler translating SIDL into a conventional programming language
  - support for Java, C++, Python, Fortran
  - objects can be distributed (use RMI)
Typical Workflow

user

expresses object interfaces and contracts in SIDL

Babel

artifacts

Kosta Damevski - “Parallel Object Contracts for HPC”
Typical Workflow

- User expresses object interfaces and contracts in SIDL
- Babel compiles SIDL and generates code
- Artifacts

Kosta Damevski - “Parallel Object Contracts for HPC”
Typical Workflow

user
expresses object interfaces and contracts in SIDL

Babel
compiles SIDL and generates code

artifacts

Kosta Damevski - “Parallel Object Contracts for HPC”
Typical Workflow

user

expresses object interfaces and contracts in SIDL

Babel

compiles SIDL and generates code

artifacts

glue code

method skeletons

Kosta Damevski - “Parallel Object Contracts for HPC”
Typical Workflow

user
expresses object interfaces and contracts in SIDL

Babel
compiles SIDL and generates code

artifacts

[glue code]
[method skeletons]

Kosta Damevski - “Parallel Object Contracts for HPC”
Typical Workflow

user
expresses object interfaces and contracts in SIDL
writes implementation in skeletons

Babel
compiles SIDL and generates code

artifacts
+ glue code
+ method skeletons

Kosta Damevski - “Parallel Object Contracts for HPC”
bool vuIsZero(in array<double> u, in double tol)
require
  not_null : u != null;
  u_is_1d : dimen(u) == 1;
  non_negative_tolerance: tol >= 0.0;
ensure
  no_side_effects : is pure;

• SIDL extensions for expressing contracts as method preconditions, postconditions and class invariants
• Support for both simple and complex contracts
  • Logical operations (and, or, implies, if and only if, etc.)
  • Contracts can make function calls

Kosta Damevski - “Parallel Object Contracts for HPC”
Concurrent vs. Parallel Objects

Concurrent Object
(one instance)

Parallel Object
(multiple instances)
Concurrent vs. Parallel Objects

Concurrent Object (one instance)

Parallel Object (multiple instances)
Concurrent vs. Parallel Objects

our work focuses on parallel objects

Kosta Damevski - “Parallel Object Contracts for HPC”
Parallel Objects

- Their implementation contains calls to MPI, PVM or some other such library
- All of the processes/instances can be identified as a group
  - use the same communicator
- Babel does minimal work to make this happen
Talk Outline

• Introduction
• Background
• Proposed Parallel Contract Clauses
  • Data Parallelism Clauses
  • Task Parallelism Clauses
• Preliminary Evaluation
Proposed Data Parallelism Clauses

```cpp
bool vuIsZero(in array<double> u, in double tol)
sync-begin
  is collective;
  numThreads > 1;
```

- **New clauses** `sync-begin` and `sync-end` indicate the time of enforcement
- Enforced before or after the method invocation
- **New contract operations specific to these causes**
  - `is collective` - ensure that all participating execution units invoke a method
  - `numThreads` - enable contracts specifying limits on the number of execution units

Kosta Damevski - “Parallel Object Contracts for HPC”
Algorithm for enforcing *is collective:*
1. choose a master (node 2)
2. all other threads send an async message to master
3. if master receives all messages, then it is collective
4. if master times out, then throw an exception
User-defined Assertions

• Any SIDL defined function can be used in a parallel contract, as long as the function is:
  • side-effect free (i.e. contains a is pure assertion)
  • static, or belongs to the same class
    • no way to use an object instance inside of a contract
• Babel will make sure that is collective and contracts containing numThreads would be enforced first
  • ensuring that the intra-contract function is called in the same conditions as the parent function
Proposed Task Parallelism Clauses

parallel class Matrix {
    double matNorm(in double tol)
        sequence
            after matInit;
            before matDestroy;
    ...
}

• sequence clause sets an invocation ordering between a number of methods

• simple sequences ensure a loose ordering between the methods

• other methods of the class can be called between matInit and matNorm and matNorm and matDestroy

Kosta Damevski - “Parallel Object Contracts for HPC”
Strict Sequences

- strict sequences enforce a mandatory order of invocation
- matInit must follow matAllocate or the contract is violated
Global Sequences

parallel class Matrix {
  double enable()
  sequence
    global after disable;
  ...
}

- global sequences apply across all of the parallel instances of a class
- each invocation of enable on any instance must be followed by a disable on some instance
- not many good examples so far of when these would be useful
- Only one added in our current studies

Kosta Damevski - “Parallel Object Contracts for HPC”
Contract Violations

- Contract violations result in an exception
- Difficult to express exceptions in a parallel scenario
  - The exception occurs at only one process/thread
- Previous work shows how to perform the distribution of exceptions in this setting
  - Synchronously
  - Asynchronously - but exceptions may arrive with a delay
Talk Outline

- Introduction
- Background
- Proposed Parallel Contract Clauses
- Preliminary Evaluation
  - SPH
  - Hypre
Study I: SPH

- Smoothed Particle Hydrodynamics (SPH) code used for groundwater simulation at PNL
- Consists of
  - a dozen CCA components
  - 29 methods
- We added 48 sequence and 5 synchronization contract clauses to Lagrange component

Kosta Damevski - “Parallel Object Contracts for HPC”
Study 1: SPH

- Smoothed Particle Hydrodynamics (SPH) code used for groundwater simulation at PNL.

Major obstacle was that in places, the code was not fully decomposed into objects which made some contracts impossible to express.

- 29 methods
- We added 48 sequence and 5 synchronization contract clauses to Lagrange component.
Study II : Hypre (LLNL)

- Linear algebra package for both structured and unstructured grids
- Commonly used and a good candidate for contracts that may find errors in its use
- We instrumented:
  - 49 classes containing 149 methods
  - 24 sequence clauses, 9 synchronization clauses
Study II : Hypre (LLNL)

• Linear algebra package for both structured and unstructured grids

Major obstacle was the current inability to express contracts using object instances

• We instrumented:
  • 49 classes containing 149 methods
  • 24 sequence clauses, 9 synchronization clauses
### Contracts Runtime Overhead

<table>
<thead>
<tr>
<th></th>
<th>Percent of App. Time In Contracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPH</td>
<td>0.004%</td>
</tr>
<tr>
<td>Hypre</td>
<td>1.136%</td>
</tr>
</tbody>
</table>

- Overhead of contracts in an important consideration
- Remains low in the cases we have considered
Sample of the Related Work

- A large body of work considering contracts for concurrent code (Nienaltowski, Meyer ‘06)
  - Difficult to ensure the precondition holds for the duration of the thread’s execution
- Proposals for expressing synchronization in the interface (Ling et al. ‘99)
  - Perform synchronization at the interface level
  - May lead to poor performance if locking is too coarse-grained
Summary of Contributions

• A set of contract clauses and operations for parallel object / component applications
  • enforce collective behavior, number of threads and allow user defined functions
  • provide for sequences between method invocations
Future Work

• Further evaluation
  • Consider more application examples
• Enabling non-static, instance based contracts requires careful consideration
  • Potential pitfalls, similar to those in concurrent contracts exist
Questions?

Kosta Damevski
kdamevski@vsu.edu

work sponsored by the Department of Energy and NASA