Using Model Checking and Symbolic Execution for the Verification of Data-Dependent Properties of MPI-Based Parallel Scientific Software

CRA Distributed Mentor Project Summer 2004

Mentee Anastasia Mironova, University of Alaska Anchorage

Mentors Lori A. Clarke, Stephen Siegel, George Avrunin; Laboratory for Advanced Software Engineering Research, University of Massachusetts, Amherst

Problem Description

It is hard to write “correct” parallel programs

- Concurrency adds complexity and introduces problems such as deadlock
- Non-determinacy makes testing even less effective

Model checking techniques have been applied to concurrent systems

- But focus on patterns of communication instead of correctness of the computation
- Limited experience with MPI-based programs

Approach

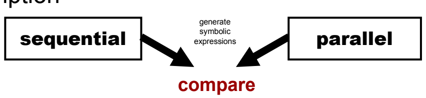
- Use a model checker to explore all possible executions
- Verification of freedom from deadlock
    - Modeling MPI functions
    - Abstracting away unnecessary data
- Verification of computational correctness
    - Extending the model checker to create symbolic representations of the executions
    - Comparing the sequential program’s symbolic representation of the result to the parallel program’s representation

Case Studies

Example 1: Multiplication of Matrices

Verification procedure

1. Symbolic computation is performed in parallel, generating a matrix of expressions on the root process
2. The root process does the symbolic computations sequentially
3. The root process loops through the two resultant structures checking that they are the exactly the same via a set of assertions description



Example 2: Gauss-Jordan Elimination

This case presents a greater challenge

- Need to introduce branching where conditions are functions of data
- The property is harder to express since there is no closed formula of the answer, again, consequence of data dependencies

We modify the verification procedure of Example 1 to match not only the symbolic expressions but also the set of path conditions.

Conclusions and Future Work

Conclusions

Deadlock

- Demonstrated applicability of abstractions
- Demonstrated scalability: ability to handle non-trivial sizes of matrices

Computational correctness

- Sequential model capable of handling non-trivial sizes of matrices
- Used the SPIN Model Checker to create symbolic expressions and compare these expressions for the parallel and sequential versions of the algorithm

Future Work

- Improving the PROMELA models
    - Optimizing data structures
    - Incorporating C code
    - Employing theorem proving packages
- Using a different model checker, e.g. MOVer (MPI-Optimized Verifier)
- Exploring other non-trivial computational examples