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

Mentee: Anastasia Mironova
University of Alaska Anchorage, Anchorage, Alaska
anastasia.mironova@gmail.com
Mentors: Lori A. Clarke, Stephen Siegel, George Avrunin
Laboratory for Advanced Software Engineering Research
University of Massachusetts at Amherst

ABSTRACT

The objective of this project was to investigate the application of model checking to the formal verification of properties of MPI-based parallel scientific programs. By extending a model checker with symbolic execution capabilities, we were able to verify that a parallel version of the system computed the same values as a sequential version. We experimentally evaluated this approach for two small numerical programs. Although more work needs to be done, this project served as a proof of concept.

1. INTRODUCTION

Problems in the field of scientific computation often require extensive parallelization of the involved algorithms in order to execute in reasonable time. Hence, the majority of code for solving such problems is written for parallel architectures. Programming for such architectures is challenging, making it even harder to write "correct" code.

Programs written for parallel architectures are not only hard to create, they are significantly harder to test. Parallelism adds complexity and introduces problems, such as deadlock and race conditions. Problems like these can be extremely difficult to detect. The non-deterministic decisions made by the runtime system can make testing even less effective as it can be difficult to reproduce the same sequence of execution. Hence, it becomes important to have other means for testing and debugging parallel code.

In this work we are investigating the application of model checking to code written for parallel architectures using a high-level language with a library of message passing routines. In particular, we are considering two common non-trivial mathematical routines for computing the product of two square matrices and Gauss-Jordan elimination. The original code is written in C using the popular Message-Passing Interface (MPI) library. The two specific properties we are concerned with here are freedom from deadlock and computational correctness and the particular model checker we are using to carry out the verification step is the SPIN Model Checker.

This work is intended to contribute to the current knowledge in this area by expanding on several aspects. Model checking techniques have been applied to concurrent systems in the past but their focus has been primarily on patterns of communication rather than the correctness of the computation. Past studies exhibit limited experience with MPI and, hence, this work is a contribution in this direction as well.

2. APPROACH

The choice of using model checking directs us to using the following approach for carrying out the formal verification.

We construct abstract models of the original C code and use a model checker to explore all possible executions. A separate model is built for each particular property in order to maximize the degree of optimization. The models are carefully constructed to accurately represent all components of the original code relevant to the specific property being checked and abstract away most of the irrelevant details.

Once the models are constructed to represent the relevant components of the original code, we define a set of conditions that describe the property we are checking and make modifications to the models to contain the corresponding checks for these conditions. For checking freedom from deadlock there is no need to explicitly include the description of this property in the code because this check is performed automatically in SPIN. The situation is, however, less trivial for specifying the correctness of the computation property.

The property of computational correctness is hard to express in general, so we must first define what it is that we believe to be correct. Since in this work we are concerned with correctness of the parallel code, we base our definition of computational correctness on the assumption that the sequential version of the same algorithm is correct. Hence, in order to check correctness of computation of the parallel algorithm we must verify that the parallel version of the algorithm is equivalent to sequential, that is for every possible execution and for all possible values of the input data both versions of the algorithm produce equivalent symbolic expressions. As it turns out, the specific set of assertions that must be evaluated by the model checker to verify this property varies depending on the computational example.

3. CASE STUDY

In this work the two examples of non-trivial mathematical computation we are considering are multiplication of two square matrices and Gauss-Jordan Elimination. For the case of multiplication of matrices we construct two models, one specific to verifying freedom from deadlock and the second for checking the correctness of computed expressions. For the second example Gauss-Jordan elimination, we restrict our attention to only the property of computational correctness. The following two sections describe these case studies in detail.

3.1 MULTIPLICATION OF MATRICES

This example was the first one considered during the course of the project and the original C code was adopted from an assignment written for an undergraduate course in programming languages at the University of Alaska Anchorage. This code performs multiplication of two square matrices in parallel. For this example we constructed two models, one for checking freedom from deadlock and the other one for verifying correctness of computation. We will now discuss the specificities of abstractions and verification procedures for each model.

For verification of freedom from deadlock the most relevant component of the original code that we model is the MPI functions so we could abstract away much of the detail about the input data and computation. The verification step in this case is performed by simply running the model checker on the constructed model in verification mode. SPIN is set to automatically check for deadlock by default.

For verification of computational correctness the key modeling decision is the use of symbolic expressions that we construct in order to represent numerical data and mathematical expressions involved in the process of computation. We extend the model checker to create these symbolic representations of the executions and carry out any necessary manipulations.

Verifying the computational correctness property in this model is slightly more complicated. The following list outlines the sequence of steps we perform in order to carry out the formal verification:

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 exactly the same via a set of assertions description.

The assertion statements in SPIN are also checked automatically when this model checker is run in the verification mode, and hence that is how we carry out the verification step here.

3.2 GAUSS-JORDAN ELIMINATION

The Gauss-Jordan Elimination algorithm defines a sequence of elementary row operations to reduce any matrix to its reduced row echelon form. Gauss-Jordan Elimination is a very standard computational routine and for this example we are only verifying the property of computational correctness.

As it turns out, this is a much harder problem compared to the previous example of multiplication of matrices. There are two main differences. First, to accurately model the program to take into account all possible values of the input data, we must model branching, where conditions are functions of the input data. Second, the property of computational correctness in this case is harder to express since there is no closed formula of the answer, which is, again, a consequence of conditional data dependencies.

To handle the above problems we modify the verification procedure defined earlier for the matrix multiplication example in the following manner:

1. Symbolic computation is performed sequentially, generating a matrix of expressions on the root process as well as maintaining a set of path conditions that has been followed during the execution
2. The computation is performed in parallel following the set of path conditions generated at the preceding step
3. The set of processes which participated in the parallel computation assert that their symbolic representation of the result matches the corresponding part of the sequential computation of Step 1.

This procedure ensures that under the assumption that the sequential implementation is correct the parallel version of the same algorithm will also produce correct results for every possible execution.

4. CONCLUSIONS AND FUTURE WORK

The work on this project is still in progress and more experimental data are still being collected. The initial results are promising, however. For verification of freedom from deadlock on the considered algorithm we have demonstrated applicability of abstractions and reasonable scalability. The constructed models were capable of performing verification on some examples of non-trivial cases of matrices.

For verification of computational correctness we have also observed reasonable scalability of the models to handle non-trivial dimensions of matrices. We have also demonstrated the ability to employ the SPIN Model Checker to create and manipulate symbolic expressions and compare these expressions for the parallel and sequential versions of the algorithm.

The future work for this project has three main directions: improving the existing SPIN models, using a different model checker, and exploring other non-trivial computational examples. The first direction can also be subdivided into three categories: optimization of existing data structures, incorporating fragments of C code into the actual models, and possibly employing theorem proving packages to assist with manipulation of the symbolic expressions.

5. REFERENCES

[1] Gropp, W., Huss-Lederman, S., Lumsdaine, A., Lusk, E., Nitzberg, B., Saphir, W., Snir, M.: MPI - The Complete Reference: Volume 2, The MPI Extensions. MIT Press, Cambridge, MA (1998)

[2] Holzmann G.: The SPIN Model Checker. Primer and Reference Manual. Addison Wesley, 2003

[3] Howard Anton: Elementary Linear Algebra. Wiley, 1977, Section 1.2

[4] Stephen F. Siegel, George S. Avrunin, Modeling MPI Programs for Verification, Department of Computer Science, University of Massachusetts, Amherst, MA 01003, September 2004. (UM-CS-2004-075)

[5] Snir, M. Otto, S., Huss-Lederman, S., Walker, D., Dongarra, J.: MPI- The Complete Reference: Volume 1, The MPI Core. 2 ed. MIT Press, Cambridge, Massachusetts (2003)

[6] Wilkinson B., Allen M.: Parallel Programming. Techniques and Applications Using Networked Workstations and Parallel Co mputers. Prentice Hall, 2nd edition 11.3.