Verifying Parallel Programs with MPI-Spin

Stephen F. Siegel[⋆]

Verified Software Laboratory
Department of Computer and Information Sciences
University of Delaware
Newark, DE 19716, USA
siegel@cis.udel.edu
http://www.cis.udel.edu/~siegel

Standard testing and debugging techniques are notoriously ineffective when applied to parallel programs, due to the numerous sources of nondeterminism arising from parallelism. MPI-Spin, an extension of the model checker Spin for verifying and debugging MPI-based parallel programs, overcomes many of the limitations associated with the standard techniques. By exploring *all possible executions* of an MPI program, MPI-Spin can conclude, for example, that a program cannot deadlock on any execution. If the program can deadlock, MPI-Spin can exhibit a *trace* showing exactly how the program fails, greatly facilitating debugging.

This tutorial will serve as an introduction to MPI-Spin. Through a series of examples and exercises, participants will learn to use MPI-Spin to check for deadlocks, race conditions, and discrepancies in the numerical computations performed by MPI programs. The only prerequisites are familiarity with C and the basic MPI operations; no prior verification experience is required. Participants are encouraged to download and install MPI-Spin before the tutorial, following the instructions at `http://vsl.cis.udel.edu/mpi-spin`.

The tutorial is divided into four parts, each lasting approximately 45 minutes: (1) introduction and tool demonstration, (2) language basics, (3) using MPI-Spin, and (4) verifying correctness of numerical computations.

1. Introduction and Demonstration. The introduction will begin with a discussion of some of the most common problems plaguing developers of MPI programs. In addition to the issues mentioned above, issues related to performance, such as the question of whether or not it is safe to remove a particular barrier statement from an MPI program, will also receive attention. The limitations of testing and other dynamic methods will also be explored.

The basic tasks involved in model checking will then be introduced: the construction of a *model* of the program being verified, the formulation of one or more *properties* of the model, and the use of automated algorithmic techniques for checking that every execution of the model satisfies the property. The limitations of model checking will also be discussed; these include the *state explosion problem* and the problem of accurately constructing appropriate models of programs. It will be emphasized that, while modeling requires a certain degree of

⋆ This material is based upon work supported by the National Science Foundation under Grant No. 0541035.

skill, it is not more difficult than programming and with a little practice most MPI programmers can become very effective modelers.

The remainder of this part of the tutorial will consist of a tool demonstration, which will also introduce the main example used throughout the tutorial, the **diffusion** program.

2. Language Basics. Some knowledge of programming languages carries over into modeling languages, but modeling differs in several significant ways. In this part of the tutorial, the basic syntax and semantics of the MPI-Spin input language will be described in a progressive, methodical way. The description will start with those syntactic elements dealing with process declaration and management, then move on to variables and types, then expressions, and finally the different types of statements provided by the language. Throughout, the **diffusion** example will be used to illustrate the various language constructs.

3. Using MPI-Spin. This part will begin with a discussion of *abstraction* and how the choice of appropriate abstractions can lead to models that are both *efficient* and *conservative*. These notions will be defined precisely and illustrated using **diffusion** and **matmat**, a program that computes the product of two matrices using a master-slave pattern. In the latter example, the consequences of using various abstractions for the variables in the program will be explored. It will be shown that different choices are appropriate for verifying different properties of **matmat**.

Once a model has been constructed, the MPI-Spin tool itself must be executed on that model. As is the case for any complex tool (such as a compiler), there are many options, parameters, and flags available to the user. The most commonly used options will be described and their effects demonstrated using the examples introduced previously.

This will be followed by a "hands-on" exercise in which MPI-Spin is used to explore the consequences of modifications to the **diffusion** code.

4. Verifying Correctness of Numerical Computations. The fourth part of the tutorial deals with a recent and exciting development in the field of model checking for HPC: techniques using symbolic execution to verify properties of the numerical computations carried out by parallel programs. These techniques are supported in MPI-Spin through an abstract datatype `MPI_Symbolic` together with a number of operations on that type, such as `SYM_add` and `SYM_multiply`. The idea is to model the inputs to the program as symbolic constants x_i and the output as a vector of symbolic expressions in the x_i. The output vector can be analyzed for a number of purposes. Most importantly, it can be compared against the symbolic output vector produced by a trusted sequential version of the program. In this way, model checking can be used to show that for *all inputs*, and for *all possible executions* of the parallel program on the input, the parallel program will produce the same results as the sequential one. This technique will be illustrated using **matmat** and, if time permits, a more complex example implementing the Gaussian elimination algorithm.