Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 193 (2007) 19–27
www.elsevier.com/locate/entcs

# Formal Methods for MPI Programs

Ganesh Gopalakrishnan and Robert M. Kirby [1,2]

*School of Computing*
*University of Utah*
*Salt Lake City, UT 84112, USA*

**Abstract**

High-end computing is universally recognized to be a strategic tool for leadership in science and technology. A significant portion of high-end computing is conducted on clusters running the Message Passing Interface (MPI) library. MPI has become the *de facto* standard in high performance computing (HPC). Our research addresses the need to avoid bugs in MPI programs through a combination of techniques ranging from the use of formal specifications to the use of *in-situ* model checking techniques. This paper details an assessment of the efficacy of these techniques, as well as our future work plans.

*Keywords:* MPI, distributed programs, message passing, formal methods, model checking, formal specifications

## 1 Introduction

The importance of high-end computing (sometimes called high performance computing, or HPC) needs very little motivation in a modern context. HPC is universally recognized to be a strategic tool for leadership in science and technology. A significant portion of high-end computing is conducted on clusters running the Message Passing Interface (MPI [1]) library. In these applications, system models occurring across a broad range of real-world applications—anywhere from chemical boilers to weather models—are studied through simulations. MPI has become the *de facto* standard in HPC. As an expert observes [2], MPI's popularity is due to several resons:

[1] Email: ganesh@cs.utah.edu
[2] Email: kirby@cs.utah.edu


doi:10.1016/j.entcs.2007.10.005

- Portability: The need for portability is high, because code tends to outlive hardware.
- The ability to smoothly map primitives to hardware architectures.
- The large variety of calls that allows each user to find subsets that are well-matched to their needs.
- Its orthogonality in terms of what combinations of features are allowed.

Just as is the case with parallel programs in general, MPI programs in particular can contain bugs. Specifically, the sources of MPI program bugs have been identified to include the following. First of all, the large number of functions in MPI libraries can overwhelm developers. Second, MPI is most commonly taught or learned at an informal level. As such, programmers writing advanced MPI applications may overlook corner cases. Finally, MPI programs are not static; they are sometimes manually re-tuned when ported to a new hardware platform.

This paper is about our ongoing research that emphasizes the use of formal methods for making the creation of bug-free MPI programs easier. Our approach consists of (i) developing a formal model of the MPI library, (ii) developing *in-situ* (run-time) model checking tools, and (iii) developing static analysis support for enhancing the efficacy of model checking. This paper briefly describes our ongoing work in these areas, as well as our future plans. The authors wish to acknowledge the impact that Professor Gary Lindstrom had in the parallel computing research conducted at the University of Utah, and thank him for his feedback and encouragement of the research reported here.

The rest of the paper is organized as follows. In Section 2, we present an overview of our work underway in developing a formal specification for MPI. In Section 3, we describe our work on developing an *in-situ* model checker for MPI. *In-situ* model checking was introduced in VeriSoft [13] in the context of directly model checking C/C++ programs. Ours is believed to be the first realization of this idea for MPI programs. In Section 4, we present an assessment of our work so far, and draw conclusions for the future.

*Related Work:*

As far as we now, there is only one other group – namely that of Siegel and Avrunin – that has actively investigated the use of formal methods for high performance computing using MPI. Some of their past publications include [7,8,9]. The main differences between these works and ours are the following: (i) in Siegel and Avrunin's work, a declarative formal specification for MPI has not been proposed; (ii) they employ traditional model checking using SPIN