Formal Analysis for Debugging and Performance Optimization of MPI

Ganesh L. Gopalakrishnan and Robert M. Kirby

School of Computing, University of Utah,
Salt Lake City, UT 84112
{ganesh, kirby}@cs.utah.edu
`http://www.cs.utah.edu/formal_verification`

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 a de facto standard in HPC. MPI programs, as well as MPI library implementations can be buggy, especially when aiming high performance, and running on or porting onto new platforms. Our recent work has addressed the following areas: A TLA+ Formal Semantics of a large subset of MPI-1; A Microsoft Phoenix based Model Extraction and Analysis Framework for MPI programs; Integration into the Visual Studio Environment for error-trace visualization; A new dynamic partial order reduction algorithm (DPOR) tailored to MPI so that the number of interleavings examined during MPI program verification are dramatically reduced; A program called 'inspector' for Analyzing C++ Programs that has found bugs in publicly distributed threaded programs (Inspector automatically instruments PThread programs and searches for races based on a new DPOR); Verified Byte-range Locking Protocols using MPI one-sided Communication - a case study where we found bugs in published byte-range locking protocols, and designed and verified improved versions of these protocols; A New In-situ Model Checker for MPI programs, that traps MPI calls using its profiling interface (PMPI) and orchestrates control to maximize coverage with minimal state saving overhead. The progress made in exploring these directions, our publications, and associated software tools are described, as are our future plans.

[1]

[1]This research was funded by NSF award CNS-0509379 and Microsoft HPC Institutes

1-4244-0910-1/07/20.00©2007 IEEE.

1. Introduction

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 a de facto standard in HPC.

It is well known that bugs do get introduced during MPI programming, both within user applications that use MPI, and unfortunately, within the implementation of MPI library functions themselves. This is true even if software development is carried out by conscientious and experienced professionals. The seriousness as well as frequency of bugs is directly proportional to the degree of performance sought. Broadly stated, our research program is aimed at developing tools and techniques for checking these assumptions at all stages of the evolution of cluster computing software, starting from high level specifications all the way to optimized programs.

The MPI library provides over 300 functions, in effect serving as an 'assembly programming notation' for parallel programming. Different subsets of these functions are relevant for specific applications. Even for a single application, programmers are known to initially employ higher level (albeit less efficient) primitives such as `MPI_Send`, and subsequently transform their programs over to employ more efficient (and often lower level) primitives such as `MPI_ISSend`. These changes can introduce subtle bugs pertaining to message completion orderings, resource (e.g., buffer) conflicts, etc., and also introduce subtle platform dependencies that render the programs non-portable. Traditional debugging approaches based on testing and event visualization have not solved these problems in a fundamentally innovative manner, causing bugs to manifest in the context of large-scale clusters, and often crashing long-running simulations on expensive machines. We are investigating numerous ways – all based on the use of *formal methods* – to avoid introducing these bugs, to detect them when inadvertently inserted, and smoothly integrate with the artifacts of current software practice. 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 (e.g., [1, 2, 3, 4]).

We had to decide on a research strategy: whether to pursue a single topic in great depth, or whether to pursue a collection of complementary approaches. In the end we decided to pursue a collection of complementary approaches, motivated by the following facts: (i) effective testing and debugging requires an arsenal of related techniques; a single technique cannot work well across a spectrum of examples; and (ii) only by trying multiple approaches would one know the innate strengths and weaknesses of each approach, and more importantly the *synergistic relationships* between the approaches.

Section 2 summarizes our recent work. Section 3 presents our proposed work during the remaining two years of our NSF project. Section 4 offers concluding remarks.

2 Summary of Recent Work

Our initial proposal included the following:

- User MPI programs will be analyzed, and its control/communication skeleton will be extracted and analyzed.
- The extracted models will be subject to formal analysis using a parallel and distributed model checking framework.
- A formal semantic definition for a significant subset of MPI will be created.
- Based on the formal semantic representation, focused tests will be developed for the MPI library.
- Particular attention will be paid to new MPI-2 features such as one-sided communication.
- Our framework will be applied to a graded series of benchmark examples.

We have made progress on the above topics and also have embarked on additional directions. Here is some context for our ongoing work:

- We have obtained some funding from the Microsoft HPC Institutes program.
- We are having very fruitful ongoing collaborations with the Argonne National Labs, especially with Rajeev Thakur and Willam Gropp. The ANL team has helped us achieve deeper results rapidly, and provided us valuable feedback at critical junctures. We also have co-authored a conference paper (which was recognized as one of the three outstanding papers at EuroPVM/MPI 2006 [5]). We have expanded the EuroPVM/MPI paper, and have submitted a journal special issue paper co-authored with the ANL collaborators.
- We are building two frameworks: one based on Microsoft Phoenix and Visual Studio, while the other is based on the GCC compilation framework. The use of the Microsoft platform (called CCS) gives us additional testing opportunities; however, CCS runs only on 64-bit machines. The use of GCC allows our code to be readily used by Unix/Linux users. The MPI libraries used in this tool-chain, namely MPICH, can run on 32-bit machines also.

We now summarize the highlights of our ongoing work in individual sections below.

2.1 Parallel/Distributed Model Checking

We have implemented a parallel and distributed model checker for a modeling language and model checker called Murphi. Murphi can be used to describe concurrent protocols, and detect bugs through model checking. Murphi is widely used, especially for modeling and verifying cache coherence protocols. However, like any other model checker, Murphi suffers from state explosion. In addition to clever algorithms that avoid generating certain states (some of which, such as symmetry detection, are already present in Murphi), the use of Murphi in practice requires high state generation rates as well as large state storage capacities.

We have developed an MPI based distributed implementation of Murphi called `Eddy_Murphi`. This tool also employs two threads per node, abiding by the `MPI_thread_funneled` model of threading supported in MPI. The use of threads renders the code modular by partitioning the concerns of state generation and message handling. Linear speedups have been obtained running on over 100 cluster nodes. `Eddy_Murphi` runs on MPICH as well as Microsoft CCS, and also proves to be a valuable benchmark example in our research. Its code, and a conference paper which appeared in SPIN 2006 [6], are downloadable from our website.

2.2 Formal Semantics of MPI-1

We have developed a formal semantics for about 30 MPI-1 functions in the modeling language TLA+ (developed by Leslie Lamport of Microsoft Research). Our formal model very precisely describes what MPI-1 functions mean. We have inserted annotation tags that cross-link virtually every clause in this formal model to the English reference specification of MPI-1, thus allowing a user to understand MPI-1 (and all its nuances) as a natural progression of details, starting from the English and going into the formal semantics on demand. The formal specification can be executed in the TLC model checker, thus allowing users to write short "litmus tests" that help check users' understanding of the MPI-1 primitives. In library based concurrent software design, such a "semantic calculator" that enhances understanding of library functions has been a coveted goal. Our formal semantic achieves this goal through parsimony of expression, executability with useful speeds, and error traces displayed in the Visual Studio environment, as described in Section 2.5. The formal semantics and associated tools will be released in the next six months, and is subject of a published paper [7] as well as some in preparation.

2.3 Phoenix-based Model Extraction and Analysis Framework

For our CCS-based framework, we abandoned our adhoc compiler front-end work following a very timely suggestion made by Dr. Shahrokh Mortazavi to use Phoenix (during the Dresden ICS 2006 meeting). We have wholeheartedly embraced this framework, which now provides a solid basis to perform program analysis, complexity reduction (e.g., through slicing and "standard compiler-like optimizations"), and last but not least "code generation." The code we generate is a TLA+ model capturing the control skeleton of the input MPI program. We can therefore accept short MPI programs and model check them using this path. The MPI functions in these programs will, in such a run, be modeled using formal TLA+ definitions mentioned in Section 2.2. Our preliminary framework was demonstrated at Supercomputing 2006, and will be released over the next year, and will be the subject of a paper being written. The Phoenix framework will be used for numerous other tasks, as described in the following sections. Some of these tasks will be aimed at effecting source to source transformations between MPI programs that replace expensive MPI calls with inexpensive ones, where the substitutions are justified in a program specific manner.

2.4 Model Checking HPC Software

Direct model checking using executable formal semantic specifications is a coveted goal, but seldom demonstrated in any useful manner for "real" programs and libraries. Such an exercise is meaningful to carry out with respect to short, but intricate program scenarios. In Section 2.6, we describe some of the complexity mitigation techniques being researched, that will allow us to significantly increase the computational speed of direct model checking using formal semantics. This will be the subject of a paper being written, and the corresponding tools will be released in the coming year. Scaling even further will require the use of a customized model checker; work along these lines is proposed in Section 3.

2.5 Error-trace Visualization

We have reasonably mastered techniques for driving the Visual Studio tool chain in order to display error traces obtained during model checking. We are planning to display error-trails generated by model checkers discussed in Section 2.4 using such an interface. This will allow designers to begin using model checking, and ultimately develop trust with its results, in a familiar debugging environment.

2.6 Partial Order Reduction for MPI

During model checking, semantically independent (*commutative*) concurrent program steps from different processes should not be interleaved in all their exponential number of combinations; without such an approach, model checking can be computationally prohibitive. While such lines of research have been pursued in many settings, the notion of when two MPI calls are independent has not been researched, barring a study by Siegel [3] in the context of a limited number of MPI calls. We have developed a much more comprehensive algorithm that runs in two steps: (i) an ample-set based forward execution that employs static partial-order reduction, and (ii) a dynamic partial order reduction technique that fills in further communication command dependencies during backtracking.[2]We are prototyping this algorithm, and are integrating it into our Phoenix-based framework. In our proposed work (Section 3), a dedicated model checker embodying this algorithm will be developed. A paper is under preparation.

2.7 Our GCC/MPICH Framework

Employing a GCC/CCS based framework (*in addition* to Phoenix/CCS) is expedient for a group such as ours. We have developed such a framework that intercepts the GCC compilation flow after MPI programs have been type-checked but before they are turned into low-level three-address instructions. This framework supports many traditional high-level compiler optimizations, and includes a growing number of static analysis methods including escape analysis and alias analysis. We have begun using this framework actively in our research, including automatic instrumentation to support dynamic partial order reduction (described in Section 2.8). This framework will soon be employed to support numerous other tasks. Two examples of what we plan to do with this framework in our future work (Section 3) are: (i) static analysis methods that determine which MPI barriers can be safely removed (to increase performance), and (ii) analysis methods that determine where new MPI barriers can be inserted (for example, to streamline accesses to devices such as network adapters in order to reduce contention, to reduce the complexity of model checking, etc).

2.8 An In-situ Model Checker

Debugging threaded applications is notoriously hard, and is receiving considerable attention. Although MPI is a distributed memory programming model, threading issues are essential to address in any software endeavor in the modern context, considering the ubiquity of multicore processors, and the ongoing escalation in the use of thread programming. In case of MPI, (i) MPI library functions may be implemented by employing multiple concurrent threads that quite naturally help exploit multiple CPU cores that will be found in virtually all future computing platforms. For instance, computations of the progress engine may be carried out on CPU cores that are distinct from the cores that carry out the application program executions; (ii) MPI has defined four threading levels for applications to exploit. Under these levels, application threads may make various extents of concurrent MPI calls.

We have developed a methodology for model checking multithreaded C/C++ programs using dynamic partial order reduction to cut down the number of interleavings examined. In our current implementation of this method, we instrument (using our GCC-based framework) user level programs to be model checked. This allows a scheduler (which our methodology provides) to take control over how the interleavings among multiple MPI processes will be allowed to happen during run-time. Initial results using this approach encouraging in terms of the run-time as well as coverage obtained by this model checker. The main benefits of this approach over traditional software model checking approaches lies in the fact that the laborious and error-prone phase of extracting a formal model from user programs, and representing these models in the language of a model checker are eliminated. In-situ model checking avoids these problems, and records only the kinds of MPI calls being made and the arguments of these calls. Of course, this style of in-situ model checking has its limitations, including having to depth-bound the state space search, and not being able to check properties with full precision. However, traditional model checking approaches that rely on the extraction of formal models are incapable of being applied, and so the reduced precision is perceived to be a good compromise.

Three papers based on our in-situ model checker will be written in 2007. A tool release will follow these publications.

2.9 Verified Byte-range Locking

Often, processes must acquire exclusive access to a range of bytes, such as a portion of a file. In [8], Thakur et al. presented an algorithm by which processes can coordinate among themselves to acquire byte-range locks, without a central lock-granting server. The algorithm uses MPI one-sided communication with passive-target synchronization (`MPI_Win_lock` and `MPI_Win_unlock`). In a paper in EuroPVM/MPI 2006 that won an Outstanding Paper award, we showed that their algorithm, while ingenious, can suffer from a serious deadlock. One correction proposed to the algorithm avoids this deadlock, but introduces a livelock that can seriously hurt performance. We proposed a second alternative algorithm that avoids the livelock also. Experimental results on a cluster of 128 CPUs indicated that the second alternative performs much better than the first corrected alternative, and even the original buggy algorithm. Model-checking using existing tools such as Promela/SPIN [9] was employed. A new in-situ model checking approach for MPI processes (described in Section 2.10) finds these errors without the effort of manual modeling. Promela models for these byte-range algorithms are available from our website.

2.10 In-situ Model Checker for MPI

Very similar to the in-situ model checking approach described for threaded programs, directly model checking MPI programs can have significant advantages, including avoiding the huge burden of model extraction. MPI program executions can be intercepted using the PMPI (profiling MPI) mechanism. We have developed a scheduler that interleaves MPI process execution steps, and helps cover execution states more systematically than random testing (which tends to get locked into various execution cycles as dictated by the inherent timings of the system). Using our preliminary in-situ model checker, we have been able to locate the deadlock in the byte-range locking protocol mentioned in Section 2.9. A paper is under preparation, and software releases are planned to occur in mid-2007.

2.11 Formal Semantics Based Testing

While MPI libraries similar to CCS have been derived from mature libraries such as MPICH, a number of adaptations and performance enhancements have, nevertheless, been performed. These are potentially sources of new bugs. In addition, rapid advances in cluster hardware technology, including new interconnects, will require the MPI library functions and their supporting devices to be constantly evolved. All this emphasizes the need to be continually testing MPI library implementations. The Argonne group with whom we collaborate are sure to be pioneering many of these advances, as they have done in the past. Our collaborations with them put us in a strong position to be working on *formal testing* of MPI libraries, driven by real driving problems. Current testing methods and test suites for MPI libraries are quite inadequate. To give an example (learned from our ANL collaborators), there are only about six tests for the threading aspects of MPI in the MPICH distribution (considered the state of the art in many ways). Moreover, these tests are derived almost directly from user bug reports! The currently popular Intel/DARPA test suite applies only to MPI-1, and hence cannot be used to test the new additions in MPI-2 - precisely where bugs are more likely, because of the many extensions in MPI-2 over MPI-1. While the tests within the Intel/DARPA test suite are constructed with considerable designer insight, with the availability of a formal semantic definition for MPI, the possibility of conducting *formal model based testing* is attractive. We plan to develop this capability in our work, as elaborated in Section 3.

2.12 Benchmark Examples

We have many active projects in-house that can supply us with a graded series of benchmarks. We have also ported all the examples in the popular book by Pacheco on MPI to run on CCS. The byte-range locking protocol served as an extremely valuable real-world example. We will continue to develop more such realistic examples, thanks to the numerous collaborations (including with ANL) we have developed.

2.13 Demo at Supercomputing 2006

At Supercomputing 2006, we discussed the development of a formal semantics for MPI (Section 2.2), the use of the Phoenix framework (Section 2.3), and demonstrated the use of direct model checking based on the MPI formal semantics (Section 2.4).

3 Proposed Work

The following work is planned:

Formal Semantics of MPI-1: New work planned in this area includes the addition of communicators, the one-sided communication construct, as well as threading models. The litmus-testing capability that exists in our Phoenix-based framework will be extended in response to these additions.

Phoenix-based Framework: Considerably more features will be added to our existing Phoenix framework in order to help engineers drive static analysis tools, model checking tools, and debugging tools in a seamless manner.

Dedicated Model Checkers: Dedicated model checkers that directly implement our partial order reduction algorithms will be developed and integrated into our framework.

Visual Studio Integration: We will work closely with Microsoft's HPC researchers and engineers to best approach this integration.

Reduction Methods: In addition to cluster-based partial order reduction, other reduction methods (e.g., banking on the relative data independence exhibited by many MPI programs, the high degrees of symmetry available in SPMD programs, etc.) will be investigated and integrated.

In-situ Model Checkers: The in-situ model checkers will be made available in our framework, after further developing them in isolation to enhance their capacities.

Testing and Threading Issues: Heavy emphasis will be placed on developing testing methods that directly stem from our work so far. Even without considerable new work, it would be interesting to turn what we have into testing opportunities, simply because (i) at the end of the day, good testing methods are simply indispensable, and (ii) every new test is another opportunity to catch another class of bugs without costing much human time (machines are relatively inexpensive; expert humans are not).

4 Concluding Remarks

This paper provided a summary of our ongoing work in the area of formal methods applied to modeling MPI and thread-based programs. We described two frameworks for model extraction and verification, one based on Microsoft's Phoenix, while the other is based on GCC. We described two in-situ model checking methods, one for MPI run-time model checking, and the other for thread-level model checking. These stateless search techniques are based on dynamic partial order reduction to minimize the number of interleavings examined. Case studies conducted on published MPI protocols have enabled us to demonstrate the capability of model checking to detect errors, and eventually lead to the discovery of correct protocols that are also correct.

We have outlined an ambitious list of things to achieve in the coming two years. We plan to look into MPI 1-sided communication based implementations of high performance distributed algorithms as case studies to examine using our tools. Our collaborations with Argonne (Thakur and Gropp) will continue, and new collaborations

(Lawrence Livermore - de Supinski) are anticipated. The verification tools mentioned here are expected to be released towards the end of 2007.

PI Gopalakrishnan organized a successful workshop "Thread Verification (TV06)" in Seattle, as part of the FLoC conference series. The URL of this workshop kept at [10] contains its entire proceedings, and presents both contributed talks and invited talks. A special issue of papers selected from this workshop is being prepared for the ENTCS journal. The TV workshop series will be continued, with another event planned for the end of 2007.

A complete list of publications from this award are [5, 6, 7] which have appeared, one special issue journal paper based on [6] under review by the journal STTT, one invited journal paper based on [5] under review by the journal Parallel Computing, a special issue of selected papers from the TV06 workshop [10] being guest-edited by the PI and accepted by the ENTCS journal.

We expect the four PhD students working on this grant (supplemented also by Microsoft funding) to graduate shortly: Palmer to graduate in 2007, Yang in 2008, and Sarvani and Subodh beyond that. MS student Pervez will also graduate in 2007. Our two undergraduates Sawaya and DeLisi are expected to graduate in 2007 as well.

References

[1] Stephen F. Siegel and George Avrunin. Analysis of MPI programs. Technical Report UM-CS-2003-036, Department of Computer Science, University of Massachusetts Amherst, 2003.

[2] Stephen F. Siegel and George Avrunin. Verification of MPI-based software for scientific computation. In *Proceedings of the 11th International SPIN Workshop on Model Checking Software*, pages 286–303, April 2004. LNCS volume 2989.

[3] Stephen F. Siegel. Efficient verification of halting properties for MPI programs with wildcard receives. In *Proceedings of the 6th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*, pages 413–429, 2005.

[4] Stephen F. Siegel. Model checking nonblocking MPI programs, January 2007.

[5] Salman Pervez, Ganesh Gopalakrishnan, Robert M. Kirby, Rajeev Thakur, and William Gropp. Formal verification of programs that use MPI one-sided communication. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface (EuroPVM/MPI), LNCS 4192*, pages 30–39, 2006. Outstanding Paper.

[6] Igor Melatti, Robert Palmer, Geof Sawaya, Yu Yang, Robert M. Kirby, and Ganesh Gopalakrishnan. Parallel *and* distributed model checking in Eddy. In *Model Checking Software (13th International SPIN Workshop)*, pages 108–125, 2006. LNCS 3925.

[7] Robert Palmer, Ganesh Gopalakrishnan, and Mike Kirby. Formal specification and verification using +CAL: An experience report. In *Verify'06 workshop (Post FLoC 2006)*, 2006. TLA+ Semantic Document available from rpalmer@cs.utah.edu.

[8] Rajeev Thakur, Robert Ross, and Robert Latham. Implementing byte-range locks using MPI one-sided communication. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface, 12th European PVM/MPI Users' Group Meeting*, pages 120–129. LNCS 3666, Springer, September 2005.

[9] G. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, May 1997.

[10] Threads verification 2006 (tv06) workshop proceedings, 2006. Post FLoC Workshop - http://www.cs.utah.edu/tv06.