Analysis and Verification of Message Passing based Parallel Programs

Dhriti Khanna

IIIT Delhi, India
dhritik@iiitd.ac.in
Ph.D. co-advisors: Dr. Rahul Purandare, IIIT Delhi; Dr. Subodh Sharma, IIT Delhi

1 Introduction and Motivation

Many complex applications which have to tackle with the simulation tasks and data analysis are prerogative to High-Performance Computing. And so, high data volumes and performance requirements have held the parallel community developers with a clasp to deliver scalable, accurate, and most importantly *bug-free* solutions. Writing correct and bug-free parallel programs is harder than sequential programs because the participating entities interact in such non-deterministic ways which leads to an exponentially huge number of schedule scenarios that are difficult to anticipate and visualize before-hand. Message Passing (MP) is a prominent programming model via which nodes of a distributed system communicate. In distributed MP based programs, the communication races leading to data corruption or communication deadlocks assume importance. Programmers have to predict messaging patterns, perform data marshaling and compute locations for coordination in order to design correct and efficient programs. Unfortunately, there is a shortage of verification and formal-method techniques that can guarantee the development of correct solutions.

1.1 Research Problem

This thesis aims to provide efficient solutions to the problem of detection and debugging of errors manifested because of the parallel paradigm. Discovering the errors in MP parallel programs amounts to discovering unsafe communication structure of the programs. The work in this dissertation is instantiated for the pioneer API in MP programming model: Message Passing Interface (MPI). MPI Programs are hard to reason about correctness because of non-determinism introduced in the programs by wildcard receive calls and the different buffering facilities provided by different library implementations as MPI standard does not mandate that standard blocking sends will always block in the runtime.
Relevance: (i) Programmer productivity can be increased through formal verification tools which can prove the correctness of programs precisely, handle problems of scale, and enhance coverage by avoiding redundant searches. (ii) There are many emerging MP based technologies like MCAPI[1] (fastest grow-

[1] www.multicore-association.org/workgroup/mcapi.php

ing industry standard MP library) that will benefit from the techniques and approaches developed to verify MPI applications.

2 Literature and Research Challenges

Last few years have seen the development of variety of verification tools for MPI applications employing different concepts. Traditional error-checking and debugging tools are simply not equipped to handle parallel programming model. Tools based on static model checking [5] and explicit-state runtime model checking [8] are exhaustive but not scalable. Symbolic or predictive analyses which concertize one execution and predict the behaviors of other executions [3] do address the problem of interleaving explosion by symbolically encoding all the schedules simultaneously, but the technique is a trace verification technique and can not handle programs with multiple traces depending on the non-determinism in communication.

Static Analysis, although leveraged extensively for multi-threaded shared memory programs, remains a lesser scratched surface in the context of MPI programs. This is primarily because the number of processes is an unknown parameter before execution. Other reasons are mentioned in Section 4. It will be interesting to see how MPI programs can be statically analyzed and how the static analysis can be utilized to solve bigger problems.

3 Research Progress

As a first step towards this thesis, we have completed the following work which aims to outskirt the inadequacy of single-path trace verifiers [3] and non-scalability of runtime verifiers [8]. We evaluated our prototype tool on benchmarks from FEVS test suite [6] and from the previous research. Results show that the tool performs drastically better than the state-of-the-art runtime verifiers.

Dynamic Symbolic Verification of MPI Programs

In this work, we provide a hybrid verification technique for discovering deadlocks in *multi-path* message passing programs, *i.e.* those programs where the communicated message contents sent to a non-deterministic receiver affect the control flow of the program and the further communications. The technique is sound and complete under a fixed input and a given number of processes. We combine dynamic verification with symbolic analysis to exhaustively explore the executions of the program as follows: (i) obtain a concrete run ρ of the program via dynamic analysis; (ii) encode symbolically the set of *feasible runs* obtained from the same set of events and same control-flow branches as observed in ρ. An SMT solver is used to solve the formula encoding all interleavings corresponding with this control path; (iii) check for violations of the deadlock absence property; and (iv) if no property is violated, then alter the symbolic encoding to explore the

feasibility of taking an alternate control flow behavior which is different from ρ and initiate a different concrete run in case of such a feasibility. The work has been accepted in FM 2018.

4 Future Work

Towards the goal of helping developers create reliable software, we targeted the analysis of multi-path MPI programs for deadlock absence with better scalability. Following is the list of directions we will like to explore further:

1. *Combining Symbolic Execution and Predictive Analysis:* We have already attempted to cover non-determinism aspect of the MPI programs for the detection of deadlocks. It will be interesting to see how the predictive analysis approach can be combined with symbolic execution on the whole program to provide input coverage and to verify the program not just for a single input but under any input. Achieving full coverage of the state space biases the search towards the beginning of the space. For this problem, symbolic execution can be combined with certain heuristic-based search methods to limit the exploration space of the tool. Given a limited amount of verification time, this can greatly prune the search space without compromising on code coverage.

2. Sequential programs have been the target of many compiler analyses for optimizations and performance benefits. Many static race detection techniques have been worked upon for verifying properties for multi-threaded programs [2,4]. However, there has been a dearth of static analysis techniques for MPI programs. This is because of the following reasons: (i) The number of processes can not be known at compile time. Moreover, the number of processes can be unbounded. (ii) The non-deterministic communication primitives make it hard to analyze the program before execution. (iii) The arithmetic expressions to specify process ranks in the communication calls can be complex. In the past, only a few works have tried to extract CFG out of an MPI application code partially [1, 7]. We want to develop some methodology for static analysis of MPI, especially for the following purposes:

   (i) *Hybrid Program Analysis for Deadlock Avoidance:* Since dynamic analysis is inherently unsound, we think a wide variety of errors and some kind of deadlocks can be found by static analysis of a CFG specific to MPI. This will come at a cost of some definite false positives. Instead of putting efforts on deciding whether a defect reported by the static analysis is genuine or not (and then repairing), it will be interesting to see if it can be decided at run-time if the error will actually be occurring and then mitigating it. The idea is to develop some *restrictive* logic offline which is based on the results of the static analysis. This can then be instrumented at appropriate places in the code so as to avoid the situations arising due to property violations. In the field of multi-threaded programming, deadlock avoidance has been

considered positively [9, 10]. This approach will reduce the burden from developers to code keeping the error situations in mind. They will be able to focus solely on the problem to be solved and to code the application with maximum performance optimizations.

(ii) *Hybrid Program Analysis for State Space Reduction:* Results of static analysis can also be used to find the culprit wildcard receives for communication races and deadlocks and which must be examined at runtime. This information about such wildcard receives can be further fed into a dynamic scheduler to focus on property violations only in interleavings involving these events. This will help prune out a large state space of the problem.

Acknowledgements

The author is thankful to her PhD advisors, collaborators, and reviewers for their invaluable feedback and time. This research work is supported by TCS Fellowship awarded to the author.

References

1. Bronevetsky, G.: Communication-sensitive static dataflow for parallel message passing applications. In: CGO. (2009) 1–12
2. Engler, D., Ashcraft, K.: Racerx: Effective, static detection of race conditions and deadlocks. In: SOSP. (2003) 237–252
3. Forejt, V., Joshi, S., Kroening, D., Narayanaswamy, G., Sharma, S.: Precise predictive analysis for discovering communication deadlocks in MPI programs. TOPLAS **39**(4) (2017) 15:1–15:27
4. Naik, M., Aiken, A., Whaley, J.: Effective static race detection for java. In: PLDI. (2006) 308–319
5. Siegel, S.F.: Verifying parallel programs with MPI-spin. In: EuroMPI User's Group Meeting. (2007) 13–14
6. Siegel, S.F., Zirkel, T.K.: Fevs: A functional equivalence verification suite for high-performance scientific computing. Mathematics in Computer Science (Dec 2011) 427–435
7. Strout, M.M., Kreaseck, B., Hovland, P.D.: Data-flow analysis for MPI programs. In: ICPP. (2006) 175–184
8. Vakkalanka, S., Gopalakrishnan, G., Kirby, R.M.: Dynamic verification of MPI programs with reductions in presence of split operations and relaxed orderings. In: CAV. (2008) 66–79
9. Wang, Y., Kelly, T., Kudlur, M., Lafortune, S., Mahlke, S.A.: Gadara: Dynamic deadlock avoidance for multithreaded programs. In: OSDI. (2008) 281–294
10. Zhang, L., Wang, C.: Runtime prevention of concurrency related type-state violations in multithreaded applications. In: ISSTA. (2014) 1–12