Static Analysis of MPI Programs Targeting Parallel Properties

Sriram Aananthakrishnan
Ganesh Gopalakrishnan (advisor)
University of Utah
sriram@cs.utah.edu, ganesh@cs.utah.edu

Greg Bronevetsky (advisor)
Lawrence Livermore National Laboratory
bronevetsky@llnl.gov

I. Introduction

MPI is one of the dominant programming model for writing HPC applications. Unfortunately, debugging MPI programs is hard. A majority of research to analyze MPI programs has been focused on dynamic runtime techniques whereas, in comparison, very little work has been done on static analysis. It is well known that static analysis discovers provably true properties useful for code transformations and correctness checking. Conventional techniques lack the ability to recognize parallel properties of MPI programs. Parallel properties are properties about communication topology or properties that depend on communication; e.g., reaching constants over MPI operations. Parallel properties are valuable in (i) optimizing communication where a compiler can detect and replace collective patterns with appropriate collective call [1]. (ii) program correctness where a model checker [2] can exploit the communication information to prune its search space.

The fundamental challenge in reasoning about parallel properties is statically determining the communication topology of the program i.e., identifying the communicating operations in the program and matching them according to MPI matching rules. While this problem is undecidable, analyses can compute sound approximations for it. In this work we propose (i) new abstractions and techniques to statically discover the communication topology (ii) provide a framework for writing dataflow analyses to recognize parallel properties of MPI programs.

II. Abstracting Communication Topology

Abstracting the communication topology begins with abstracting MPI operations issued by the program. A simple abstraction is to group all the send and receive operations into two equivalence classes and connect all operations in the class of sends to that of receives.

In majority of MPI programs, the target expressions in MPI operations are determined by values of rank and communicator size. Constraining the values of these target expressions permit grouping of MPI operations satisfying the constraints into corresponding equivalence classes. The send/recv equivalence classes are then matched according to MPI matching rules [3] [4].

Happens-before ordering between MPI operations allows finer refinement of the abstractions. Bronevetsky [5] refined the abstraction using happens-before ordering and constructed parallel control-flow graph (pCFG) as an approximation for the communication topology. pCFG also grouped processes into sets of equivalence classes to operate on unbounded number of processes. The send/recv equivalence classes are matched if functional composition of their symbolic abstractions on target expressions evaluate to identity. While this technique is sound, it employs a complex matching algorithm.

Key Observations, Contributions: (i) Existing static techniques approximate the complex dynamic MPI message matching algorithm. (ii) Analyzing MPI programs with unbounded number of processes adds complexity to the matching algorithm.

In our work, we relax the requirement that the analysis apply to unbounded number of processes and focus on the case where the count is known say N. An MPI program is modeled as a CFG cross-product $CFG_1 \times CFG_2 \times \cdots \times CFG_N$. We create N analysis instances one for each MPI process.

MPI operations issued by a process are abstracted by its corresponding analysis instance into equivalence classes. We use a simple abstraction that groups all operations emanating from a statement into an equivalence class. Abstract communication topology is then computed by matching the send/recv equivalence classes. Each analysis instance carry out the semantics of an MPI operation using the MPI library. Concretely, upon reaching an equivalence class i.e., the statement for `MPI_Send` or `MPI_Recv`, each analysis instance forwards the operation to MPI library which then matches the equivalence classes. The data of the message exchanged is the dataflow state which is merged at the receiver using $\sqcup$. The equivalence classes are executed at least once making this approach sound as no communication behavior is omitted. The advantages of this approach are (i) simplified matching by delegation to MPI runtime (ii) lends into development of a framework that encourages simple analysis implementations to identify parallel properties without worrying about matching MPI operations (iii) parallel implementation as each analysis instance is independent of each other.

III. Implementation

Our techniques require composition of multiple analyses. Each analysis instance is a composition of the following (i) MPI Value: assigns concrete values to ranks corresponding to each CFG_i (ii) Constant propagation: folds the values of ranks to target expressions (iii) Unreachable code: eliminates paths that cannot be reached by each process (iv) Points-to: figures out pointer relations in MPI statements (v) MPI matching: connects the send/receive equivalence classes using MPI library.

We have implemented two composition techniques (i) sequential: analyses are run one after the other (ii) tight: analyses are executed simultaneously in a lockstep. In tight composition, analyses interact at every step improving precision of each other. It lowers the execution time of analysis composition enabling us to operate on large applications. Our implementation is in the composable static analysis framework Fuse [6] of ROSE Compiler and supports large set of C++ and MPI primitives.

TABLE I.

	MPI Value	Const. Prop.	Unreach. Code	Points To	MPI Match.
Sequential	✓	✓	✓	✓	×
Tight	✓	✓	×	✓	×

IV. Implementation Status

Table I shows the status of analyses implemented and composed under the two compositional techniques. The current approach is limited by the choice of our abstraction and the limitations are (i) non-deterministic MPI operations (ii) MPI communication inside statically unknown conditionals.

V. Future Work

Our planned extensions include (i) evaluation on large benchmarks (ii) develop new abstractions and techniques to overcome aforementioned limitations.

References

[1] T. Hoefler and T. Schneider, "Runtime detection and optimization of collective communication patterns," in *PACT*, 2012.

[2] A. Vo, S. Vakkalanka, M. DeLisi, G. Gopalakrishnan, R. M. Kirby, , and R. Thakur, "Formal verification of practical mpi programs," in *Principles and Practices of Parallel Programming (PPoPP)*, pp. 261–269, 2009.

[3] D. Shires, L. Pollock, and S. Sprenkle, "Program flow graph construction for static analysis of MPI programs," *PDPTA*, 1999.

[4] M. M. Strout, B. Kreaseck, and P. D. Hovland, "Data-flow analysis for MPI programs," in *ICPP*, 2006.

[5] G. Bronevetsky, "Communication-sensitive static dataflow for parallel message passing applications," in *CGO*, 2009.

[6] G. Bronevetsky, M. Burke, S. Aananthakrishnan, J. Zhao, and V. Sarkar, "Compositional dataflow via abstract transition systems," tech. rep., Lawrence Livermore National Laboratory (LLNL), Livermore, CA, 2013.