Exascale MPI-based Program Deadlock Detection

Raed AlDhubhani[1,*], Fathy Eassa[1], and Faisal Saeed[2]

[1] Faculty of Computing and Information Technology, King Abdul-Aziz University, KSA
Faculty of Computing, Universiti Teknologi Malaysia, Malaysia

* corresponding author: raedsaeed@gmail.com

Abstract

Deadlock detection is one of the main issues of software testing in High Performance Computing (HPC) and also in exascale computing areas in the near future. Developing and testing programs for machines which have millions of cores is not an easy task. HPC program consists of thousands (or millions) of parallel processes which need to communicate with each other in the runtime. Message Passing Interface (MPI) is a standard library which provides this communication capability and it is frequently used in the HPC. Exascale programs are expected to be developed using MPI standard library. For parallel programs, deadlock is one of the expected problems. In this paper, we discussed the deadlock detection for exascale MPI-based programs where the scalability and efficiency are critical issues. The proposed method is implemented to detect and flag the processes and communication commands which are potential to cause deadlocks in a scalable and efficient manner. MPI benchmark programs were used to test the propose method.

Keywords: Deadlock detection; Exascale systems; Message Passing Interface

1. Introduction

Exascale Computing is considered as one of the recent research topics in HPC computing area. Exascale computing refers to the capability to process 1 exaflop (10^{18} floating point operations per second). The computation capability of the current supercomputers is in the petaflops level, where 1 petaflop is equivalent to 10^{15} floating point operations per second. Manufacturing machines with this ambitious computation capability depends on using hundreds of millions of cores to achieve that computational target, which are expected to be in operation in 2020 [1]. The scientific and big data processing applications are planned to be run in these machines. So, one of the challenges is how to develop reliable applications for this parallel-based computation environment.

1.1 Message Passing Interface (MPI)

MPI is a standard library which is frequently used in the HPC. It is a standard library for HPC, which is considered by [2] as the de facto standard for parallel programming in the HPC. According to [3], MPI provides a set of functions or commands which are used by the parallel programs to facilitate the communication between the processes in the runtime. The simple scenario of using the MPI library by a parallel program is achieved by using the MPI_Send command by one process to send a message to another one in the same program, where the destination process receives the message using the MPI_Recv command. To provide a rich communication environment for the applications, MPI library provides two different types of communication: blocking and non-blocking communication. In the blocking communication, the sender and receiver must wait for the communication commands to match each other before they can proceed to execute the next instruction. In the non-blocking communication, the sender and receiver can issue the commands of the communication –MPI_Isend and MPI_Irecv- and proceed directly to execute the next command without waiting for the communication commands matching. The result of the non-blocking communication can be checked later by using the MPI_Test command to check whether the issued non-blocking command is already matched to a corresponding command or not. MPI_Wait command can also be used to enforce the process to wait for a non-blocking communication command to finish, and once that communication command is matched to an appropriate command, the process can continue its execution.

MPI provides also the wildcard receive feature MPI_Source_Any, such that the process can receive the message from any source, which leads to the message race. Because of this message race, the execution of these MPI-based programs is considered as nondeterministic, which means that the behavior of the program during the runtime may differ from one run to another.
As a result of the nondeterministic execution, MPI program testing is not an easy task. As similar to the sequential programs, the parallel programs have the same types of programming errors, like buffer overflow, division by zero, etc. In addition, the parallel programs have errors related to the concurrent communication process among their different processes.

1.2 Communication Deadlock in MPI

Communication deadlock is one of these parallel-based errors, where one process is executing a communication command –send or receive-, and this process does not find a match for that communication command, and this leads to a communication deadlock. Therefore, MPI application developers need a mechanism to detect any potential deadlock in their applications. For exascale programs which are expected to consist of millions of processes communicating with each other, detecting the deadlock requires scalable and efficient techniques. This paper presents a scalable and efficient method for communication deadlock detection for exascale MPI-based programs.

2. Related Work

In [4], In-Situ Partial (ISP) is a deadlock detection tool which is implemented by investigating all the possible interleaving in the MPI program by running the program multiple times. This tool provides complete coverage for all possible execution paths, and hence provides a guaranteed result for the deadlock detection. According to [5], this tool produces an exponential number of communication interleaving cases which makes it difficult to test MPI programs that have a large number of processes.
In [6], AND-OR Wait-For graph (WFG) is used to detect deadlocks in MPI programs. The 'AND' operation is used in this graph to represent the communication pair which is required to match each other. On the other hand, the 'OR' operation is used to represent the communication expected between sender nodes and a receiver node which has the wildcard receive feature. According to [7], using AND-OR WFG for deadlock detection is time consuming and requires high performance.
In [5], a modified version of AND-OR graph is used for the deadlock detection. Marmot Umpire Scalable Tool (MUST) provides a scalable and efficient technique for deadlock detection. However, it does not provide complete support for testing the MPI programs which have wildcard receives [8].
Model checking technique is used in [9] to detect MPI deadlocks. It explores all the possible matching and interleaving of the tested MPI program and supports the wildcard communication. The limitation of this technique is the need to construct the model of the MPI program manually.
In [10], a deadlock detection technique is suggested which does not require the testing model to be constructed manually. But, the limitation of this technique is the need to re-run the entire program many times to detect the deadlocks.
Therefore, deadlock detection in exascale MPI-based program requires an efficient and scalable technique which should be able to test millions of processes created by the program. Current deadlock detection techniques of MPI-based programs suffer from the exponential growth of the number of possible communication interleaving cases which makes the testing process of exascale MPI-based program impractical. For the other techniques which do not suffer from this exponential growth, incomplete support for the MPI communication features is provided, or the testing model construction is done manually. In both cases, deadlock detection for exascale MPI-based program cannot be achieved with these limitations.
The motivation for this research is to provide a scalable and efficient technique which does not suffer from the exponential growth of the number of possible communication interleaving cases. At the same time, the technique should have a complete support for the MPI communication features, and the ability to run the deadlock detection process without the need to construct a manual testing model.

3. Methods

This paper presents Exascale MPI-based Program Deadlock Detection (EMPDD) as a scalable and efficient method for detecting MPI deadlocks in the O(m*n) magnitude, where m is the number of processes in the program, and n is the number of communication operations in each process. EMPDD method is a static-based, and supports the wildcard receives. In addition, it investigates all the possible matching and interleaving for the MPI program communication commands.

EMPDD method consists of three algorithms: MatchProcesses, MatchOperation and DetectDeadlocks. MatchProcesses algorithm is used to apply the matching rules between the different processes of the program. To investigate the possible matching between each send command and all the potential receive commands, the MatchOperation algorithm is used. After the processes and operations matching are done, the DetectDeadlocks algorithm is used to flag all the possible deadlocks based on the produced potential matches.

In comparison to the MPI deadlock approaches which provide all interleaving cases investigation with order of exponential magnitude, EMPDD method is considered more efficient and scalable. In addition, it does not suffer from the problem of the optimized approaches which try to minimize the order of magnitude required to detect the deadlock by limiting the number of possible interleaving visited, where such limitation does not provide complete coverage to the possible execution paths.

The limitation of EMPDD method is the lack of providing a graphical notation for the execution paths which lead to the deadlock. Instead, it is capable to flag all the processes and communication commands which are responsible to produce the deadlock case. However, EMPDD method is useful to present an efficient and scalable approach to check the existence of deadlock in the MPI programs that consist of millions of processes, which is the case of the exascale MPI-based programs.

EMPDD method includes four stages: 1) extracting the MPI program communication log file 2) parsing the MPI program communication log file 3) matching the communication operations 4) deadlock detection.

The architecture of EMPDD is shown in Figure 1.

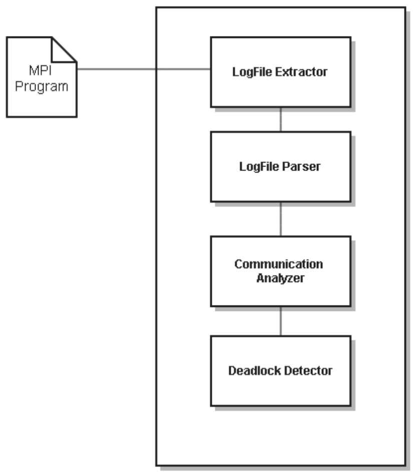

Figure 1. EMPDD Architecture

3. Results and Discussion

To evaluate EMPDD method, we used four benchmark MPI programs to check the capability of EMPDD to detect the deadlocks. The four benchmark programs are: DTG, Floyd, diffusion, and integrate programs. DTG program has a deadlock, whereas the other programs do not have any. The experimental results showed that the deadlocks are successfully detected in the DTG program. For the other programs which do not contain deadlocks, EMPDD reports them as deadlock free. Table 1 shows the result of testing the four benchmark programs, and the comparison with the ISP.

Program	Has deadlock	EMPDD Method		ISP Tool	
		Deadlock	Performance	Deadlock	Performance

		Detected		Detected	
Floyd	NO	NO	O(m*n)	NO	O(exp)
DTG	YES	YES	O(m*n)	YES	O(exp)
diffusion	NO	NO	O(m*n)	NO	O(exp)
integrate	NO	NO	O(m*n)	NO	O(exp)

Table 7. Evaluation Results

4. Conclusion

Deadlock detection for MPI programs is very important. There are many approaches which can detect the deadlocks of the MPI programs. Although some of them provide complete coverage for the possible execution paths, they are not efficient and cannot be used for complicated MPI programs. Alternative approaches solve the scalability problem and provide efficient performance to detect the MPI deadlock by limiting the number of the investigated execution paths. The cost of such limitation is the lack of the guarantee that all the execution paths are visited, and hence there is no guarantee that all the possible deadlocks are detected. In this paper, we presented an efficient and scalable method for deadlock detection in exascale MPI-based programs. The proposed method (EMPDD) is implemented to detect and flag the processes and communication commands which are potential to cause deadlocks. The limitation of this method is its lack to specify the execution paths which lead to the deadlock.

References

1. Greenough, C., Worth, D.J. and Chin, L.S. “Thoughts on Software Engineering for ExaScale Software Development,” Science and Technology Facilites Council, UK, 2011.
2. Fu, X., Chen, Z., Zhang, Y., Huang,C., Dong, W. and Wang, J. “MPISE: Symbolic Execution of MPI Programs,” Cornell University Library, Ithaca , NY, USA, 2014.
3. Message Passing Interface. [Online] www.mpi-forum.org/docs/mpi-3.0.
4. Vakkalanka, S., Sharma, S., Gopalakrishnan, G. and Kirby, R. “ISP: A Tool for Model Checking MPI Programs,” In Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming, ACM New York, NY, USA, 2008.
5. Hilbrich, T., Protze, J., Schulz, M., Supinski, B. and Müller, M. “MPI Runtime Error Detection with MUST: Advances in Deadlock Detection,” In Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, IEEE Computer Society Press, Los Alamitos, CA, USA, 2012.
6. Hilbrich, T., Supinski, B., Schulz, M. and Müller, M. “A Graph Based Approach for MPI Deadlock Detection,” In Proceedings of the International Conference on Supercomputing, ACM New York, NY, USA, 2009.
7. Deniz, E., Sen, A. and Holt, J. “Verification and Coverage of Message Passing Multicore Applications,” Journal of ACM Transactions on Design Automation of Electronic Systems, vol. 17, pp. 1-31, 2012.
8. Forejt, V., Kroening, D., Narayanaswamy, G. and Sharma, S. *Precise Predictive Analysis for Discovering Communication Deadlocks in MPI Programs*. Springer International Publishing Switzerland, pp. 263-278, 2014.
9. Siegel, S. *Model Checking Nonblocking MPI Programs*. Springer Verlag Berlin Heidelberg, pp. 44-58, 2007.
10. Vo, A., Aananthakrishnan, S., Gopalakrishnan, G., Supinski, B., Schulz, M., and Bronevetsky, G. A Scalable and Distributed Dynamic Formal Verifier for MPI Programs. In Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, 2010.