Testing and Runtime Support for MPI Applications [*]

Hongbo Li

Abstract: Over the past two decades, distributed cluster system has evolved from none to the predominant architecture in the current HPC world — it accounts for about 85% of the current top500 supercomputers. Along with the rise of cluster, MPI has evolved into the de facto standard for HPC applications on distributed clusters due to its great portability and performance. The tool support for verifying correctness of MPI applications lags far behind the ever-increasingly sophisticated hardware that has been developed. Lack of tools poses following challenges for developers: (1) there are no practical tools for systematic software testing techniques for MPI applications; and (2) there is insufficient tool support for scaling problems — a class of bugs that manifests at large scale either in terms of problem size or the number of processes.

To address the first problem, we have developed COMPI, the first concolic testing tool for MPI applications. COMPI tackles two major challenges. First, it provides an automated testing framework for MPI programs — it performs concolic execution on a single process and records branch coverage across all. Second, COMPI employs three techniques to effectively control the cost of testing as too high a cost may prevent its adoption or even make the testing infeasible.

To address the second problem, we have designed an avoidance framework for scaling problems with the use of MPI collectives. As the complexity of MPI collectives is directly impacted by both parallelism scale and problem size, their use often triggers scaling problems. Fixing a scaling problem is challenging, and thus it usually takes much time for users to wait for an official fix. To improve users' productivity, we establish the necessity of user side testing and provide a protection layer to avoid scaling problems non-intrusively, i.e., without requiring any changes to the MPI library or user programs. This provides an immediate remedy when an official fix is not readily available.

We also built a hang detection tool that saves computing resources in presence of program hangs at large scale. While program hangs on large parallel systems can be detected via the widely used timeout mechanism, it is difficult for the users to set the timeout – too small a timeout leads to high false alarm rates and too large a timeout wastes a vast amount of valuable computing resources. To address the above problems with hang detection, this paper presents ParaStack, an extremely lightweight tool to detect hangs in a timely manner with high accuracy, negligible overhead with great scalability, and without requiring the user to select a timeout value. For a detected hang, it provides direction for further analysis by telling users whether the hang is the result of an error in the computation phase or the communication phase. For a computation-error induced hang, our tool pinpoints the faulty process by excluding hundreds and thousands of other processes.

[*]The original title in the list of IPDPS'18 PhD Forum is *Correctness Support for MPI Applications.*