Tools for Simulation and Benchmark Generation at Exascale

Mahesh Lagadapati[1], Frank Mueller[1], and Christian Engelmann[2]

[1] Dept. of Computer Science, North Carolina State University, Raleigh, NC 27695-7534, mueller@cs.ncsu.edu
[2] Oak Ridge National Laboratory, engelmannc@ornl.gov

Abstract. The path to exascale high-performance computing (HPC) poses several challenges related to power, performance, resilience, productivity, programmability, data movement, and data management. Investigating the performance of parallel applications at scale on future architectures and the performance impact of different architecture choices is an important component of HPC hardware/software co-design. Simulations using models of future HPC systems and communication traces from applications running on existing HPC systems can offer an insight into the performance of future architectures. This work targets technology developed for scalable application tracing of communication events and memory profiles, but can be extended to other areas, such as I/O, control flow, and data flow. It further focuses on extreme-scale simulation of millions of Message Passing Interface (MPI) ranks using a lightweight parallel discrete event simulation (PDES) toolkit for performance evaluation. Instead of simply replaying a trace within a simulation, the approach is to generate a benchmark from it and to run this benchmark within a simulation using models to reflect the performance characteristics of future-generation HPC systems. This provides a number of benefits, such as eliminating the data intensive trace replay and enabling simulations at different scales. The presented work utilizes the ScalaTrace tool to generate scalable trace files, the ScalaBenchGen tool to generate the benchmark, and the xSim tool to run the benchmark within a simulation.

Keywords: High-Performance Computing, Message Passing, Tracing, Simulation

1 Introduction

This decade is projected to usher in the period of exascale computing with the advent of systems of up to one billion tasks and possibly as many cores. Scaling applications to such levels poses significant challenges that cannot be met with current hardware technologies. To assess the requirements for exascale hardware platforms and to gauge the potential of novel technologies, hardware simulation plays an important role in exascale projections. Significant challenges exist even at the single node level, the network interconnect and at the system level when trying to orchestrate the execution of such extensive numbers of cores as projected for exascale. Hardware simulators are vital in assessing the potential of different approaches under these challenges. Yet, these simulators need to be subjected to realistic application workloads that originate in the HPC realm. Currently, no such realistic workloads derived from large-scale HPC applications exist. This reduces simulation to studies of micro-kernels and assessment of peak metrics (bandwidth/latencies) without any notion of sustained application performance.

2 Overall Goal

This effort is targeted at alleviating the shortcomings of current hardware simulation practice by developing a universal skeleton generation capability that accurately reflects communication workloads for large-scale HPC codes.

The objective of this work is to complement the xSim simulator from Oak Ridge National Laboratory (ORNL) with benchmark generation capabilities.

To this end, the following approach has been taken:

1. ScalaBenchGen from North Carolina State University (NCSU) has been extended to auto-generate source code suitable for evaluation under xSim.
2. We have combined the ScalaBenchGen and xSim capabilities for sample HPC benchmarks/applications.

3 Our Prior Work and Related Work

Our work builds on ScalaTrace, an MPI tracing toolkit with aggressive and scalable trace compression. ScalaTrace's compression can result in trace file sizes orders of magnitude smaller than previous approaches or, in some cases, even near constant size regardless of the number of nodes or application run time [4].

ScalaTrace collects communication traces using the profiling layer of MPI (PMPI) [1] through Umpire [7] to intercept MPI calls during application execution. On each node, profiling wrappers trace all MPI functions, recording their call parameters, such as source and destination of communications, but without recording the actual message content.

ScalaTrace performs two types of compression: $intra-node$ and $inter-node$. For the intra node compression, the repetitive nature of timestep simulation in parallel scientific applications is used. Intra-node compression is performed on-the-fly within a node. Further, the inter-node merge exploits the homogeneity in behavior across different processes running the application due to the HPC-prevalent single-program-multiple-data (SPMD) programming style. Inter-node compression is performed across nodes by forming a radix tree structure among all nodes and sending all intra-node compressed traces to respective parents in the radix tree. At the parent, the respective trace representations are merged, reduced and then compressed exploiting domain-specific properties of MPI. Once propagated to the root of the radix tree, this results in a single compressed trace file capturing the entire application execution across all nodes. The compression algorithms are discussed in detail in other papers [5, 6].

As a result of these techniques, ScalaTrace produces near constant size traces by applying pattern based compression. It uses extended regular section descriptors (RSD) to record the participating nodes and parameter values of multiple calls to a single MPI routine in the source code across loop iterations and nodes in a compressed manner [2]. Power-RSDs (PRSD) recursively specify RSDs nested in a loop [3].

Another important feature of ScalaTrace is the time preservation of captured traces. Instead of recording absolute timestamps, the tool records delta time of computation duration between adjacent communication calls. During RSD formation, instead of accumulating exact delta timestamps, statistical histogram bins are utilized to concisely represent timing details across the loop. These bins are comprised of statistical timing data (minimum, maximum, average and standard deviation). ScalaTrace records histograms of delta times for each instance of a particular computation, i.e., distinguishing disjoint call paths by separate histogram instances.

We also developed ScalaExtrap, a trace extrapolation tool [9]. It contributes a set of algorithms and techniques to extrapolate a trace of a large-scale execution of an application from traces of several smaller runs. We further developed a probabilistic replay capability based on approximate matching of communication events and parameters across nodes [11]. This technique reduces trace sizes for non-SPMD codes where lossless compression techniques fail. For large-scale applications with non-SPMD or ARM-based communication patterns, such techniques could also be employed for single-node replay in the future. Another interesting direction would be to assess if receiver message content can also be replayed in a probabilistic manner for a subset of messages and, if so, how to automatically identify such messages.

Most relevant to this project is the ScalaBenchGen work [8]. It contributes an automated approach to the creation of communication benchmarks. Given an MPI application, we utilize ScalaTrace to obtain a single trace file of an HPC application run that reflects the behavior of all nodes. The trace subsequently expanded to C source code by a novel code generator. This resulting benchmark code is compact, portable, human-readable, and accurately reflects the original applications communication characteristics and runtime characteristics. Experimental results demonstrate that generated source code of benchmarks preserves both the communication patterns and the wallclock time behavior of the original application. Such automatically generated benchmarks not only shorten the transition from application development to benchmark extraction but also facilitate code obfuscation, which is essential for benchmark extraction from commercial and restricted applications.

4 Design and Implementation

We have complemented ORNL's xSim simulator with benchmark generation capabilities. To this end, ScalaBenchGen from NCSU was extended to auto-generate source code suitable for evaluation under xSim. The xSim simulator already has ample network topology support. ScalaBenchGen complements these capabilities with the ability to extract communication benchmark skeletons from actual HPC runs of applications. These skeletons include timings for computational parts and actual MPI communication calls. We have combined the ScalaBenchGen and xSim capabilities for sample HPC benchmarks/applications. Timings for the computational part have been enhanced to allow adaptation with respect to future (exascale) architectures. This co-design exploration supports the research path toward exascale.

Our initial version of ScalaBenchGen [8] is based on the first version of ScalaTrace [5] which produces lossless constant size traces for Single Program, Multiple Data (SPMD) parallel applications. ScalaTrace-2 [10] enhances the base version to achieve better compression for Multiple Program, Multiple Data (MPMD) parallel applications. ScalaTrace-2 is redesigned in every aspect such that data elements in trace are elastic and self explanatory. Because of these changes ScalaBenchGen is incompatible with the trace format produced by ScalaTrace-2. Hence, we have redesigned the ScalaBenchGen tool to generate benchmarks from traces of

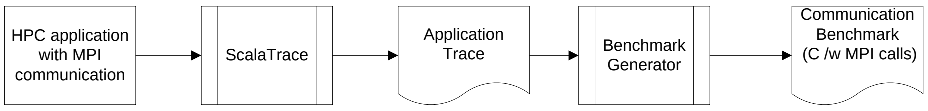
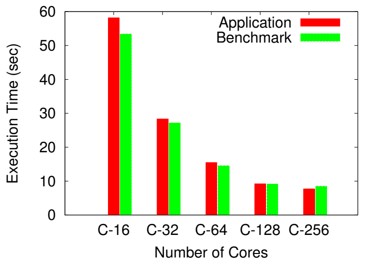
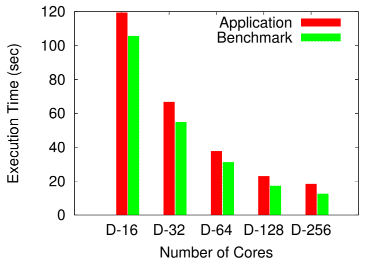

HPC application with MPI communication
ScalaTrace
Application Trace
Benchmark Generator
Communication Benchmark (C /w MPI calls)

Fig. 1. Benchmark Generation Workflow

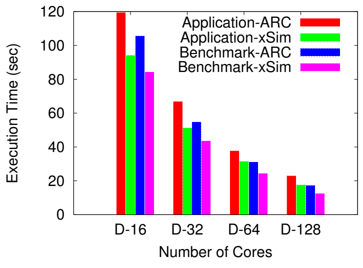
As shown in Figure 1, the application is linked with the ScalaTrace library to produce a trace file. The benchmark generator takes this trace file as an input and outputs the benchmark program. The benchmark generator can be run on a standalone machine. For every event present in trace, corresponding MPI event code is generated. Each event in the trace is reflected with its parameters and also the time elapsed between current and previous events. The benchmark generator introduces a sleep for the corresponding delta time before the event. This allows the wall clock time of the benchmark program to closely resemble that of the original application. Generated benchmarks can be combined with xSim, ORNL's extreme scale network interconnect simulator, for evaluation.



(a) FT



(b) IS



(c) IS with xSim

Fig. 2. Timing accuracy of different benchmarks

5 Early Results

ARC, a cluster with 1728 cores on 108 compute nodes, 32 GB memory per node and a QDR Infiniband Interconnect is used for evaluating our benchmark generator. Benchmarks are generated for the IS and FT codes of the NAS parallel benchmark suite (NPB v3.3). Generated benchmark runtimes are compared to the execution time of the original code's execution time.

Execution times of both the original application and the generated benchmarks are similar for the FT benchmark (see Figure 2(a)). The maximum error is 10% for FT while 30% maximum error is observed for IS (see Figure 2(b)). The higher error for IS is due to replacement of MPI_Alltoallv by MPI_Alltoall within the tracing framework, which allows a more concise trace representation (at the expense of accuracy). Figure 2(c) compares the execution times of the original code and generated benchmark on both ARC and a simulated ARC environment using xSim. Currently, xSim is not supporting a fat tree configuration, which is the topology of ARC. Hence, a fat tree is loosely approximated via a star topology. This could be the reason for the differences in observed execution times but is subject to further investigation, as is the evaluation of more NPB codes.

6 Conclusions

This work has demonstrated the capability to utilize ScalaTrace to generate concise and near lossless scalable communication traces to drive HPC archectural simulations. The resulting traces are transformed by ScalaBenchGen into a benchmark code. This code is subsequently fed into xSim to run the benchmark within a simulation environment. Ongoing work focuses on handling more benchmarks during the generation process and novel simulation techniques to handle exascale size workloads.

7 Acknowledgements

This work was supported in part by NSF grants 1217748, 0937908 and 0958311, as well, as a subcontract from ORNL. Research sponsored in part by the Laboratory Directed Research and Development Program of ORNL, managed by UT-Battelle, LLC for the U.S. Department of Energy under Contract No. De-AC05-00OR22725. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes.

References

1. MPI-2: Extensions to the message passing interface (Jul 1997)
2. Havlak, P., Kennedy, K.: An implementation of interprocedural bounded regular section analysis. IEEE Transactions on Parallel and Distributed Systems 2(3), 350–360 (Jul 1991)
3. Marathe, J., Mueller, F.: Detecting memory performance bottlenecks via binary rewriting. In: Workshop on Binary Translation (Sep 2002)
4. Noeth, M., Mueller, F., Schulz, M., de Supinski, B.R.: Scalable compression and replay of communication traces in massively parallel environments. In: International Parallel and Distributed Processing Symposium (Apr 2007)
5. Noeth, M., Mueller, F., Schulz, M., de Supinski, B.R.: Scalatrace: Scalable compression and replay of communication traces in high performance computing. Journal of Parallel Distributed Computing 69(8), 969–710 (Aug 2009)
6. Ratn, P., Mueller, F., de Supinski, B.R., Schulz, M.: Preserving time in large-scale communication traces. In: International Conference on Supercomputing. pp. 46–55 (Jun 2008)
7. Vetter, J.S., de Supinski, B.R.: Dynamic software testing of mpi applications with umpire. In: Supercomputing. p. 51 (2000)
8. Wu, X., Deshpande, V., Mueller, F.: Scalabenchgen: Auto-generation of communication benchmark traces. In: International Parallel and Distributed Processing Symposium (Apr 2012)
9. Wu, X., Mueller, F.: Scalaextrap: Trace-based communication extrapolation for spmd programs. In: ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. pp. 113–122 (Feb 2011)

10. Wu, X., Mueller, F.: Elastic and scalable tracing and accurate replay of non-deterministic events. In: International Conference on Supercomputing. pp. 59–68 (Jun 2013)
11. Wu, X., Vijayakumar, K., Mueller, F., Ma, X., Roth, P.C.: Probabilistic communication and i/o tracing with deterministic replay at scale. In: International Conference on Parallel Processing. pp. 196–205 (Sep 2011)