Auto-Generation of Communication Benchmark Traces [*]

Vivek Deshpande
Department of Computer Science
North Carolina State University
Raleigh, NC, USA
vrdeshpa@ncsu.edu

Xing Wu
Department of Computer Science
North Carolina State University
Raleigh, NC, USA
xwu3@ncsu.edu

Frank Mueller
Department of Computer Science
North Carolina State University
Raleigh, NC, USA
mueller@cs.ncsu.edu

ABSTRACT

Benchmarks are essential for evaluating HPC hardware and software for petascale machines and beyond. But benchmark creation is a tedious manual process. As a result, benchmarks tend to lag behind the development of complex scientific codes. Our work automates the creation of communication benchmarks. Given an MPI application, we utilize ScalaTrace, a lossless and scalable framework to trace communication operations and execution time while abstracting away the computations. A single trace file that reflects the behavior of all nodes is 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 application's communication characteristics and performance. Experimental results demonstrate that generated source code of benchmarks preserves both the communication patterns and the run-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.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Measurement techniques; D.1.3 [**Programming Techniques**]: Concurrent Programming—*Parallel Programming*

General Terms

Measurement, Performance

Keywords

Benchmark Generation, ScalaTrace, Performance

1. INTRODUCTION

Benchmarks are widely used for evaluating and analyzing system performance and assessing migration costs of HPC applications to platforms with different architectures. They are easy to port, modify and run, and they closely resemble the characteristics of HPC applications. But most benchmarks do not capture the complexity and scale of realistic HPC applications as they do not feature the intricate interplay of computation, communication and I/O operations.

We generate communication benchmarks automatically. These benchmarks are human readable, compact, easy to generate and port. They closely resemble the execution time and communication volume of the original application.

As an input, we take an HPC application with message passing communication using MPI (Message Passing Interface). The application's communication patterns are captured in traces. The obtained trace is given as an input to the benchmark generator, which is the central focus of this work. It outputs the communication benchmark in C code (including MPI calls for communication) that can be executed on target machine, as illustrated in Figure 1.

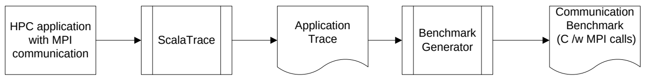

HPC application with MPI communication
ScalaTrace
Application Trace
Benchmark Generator
Communication Benchmark (C /w MPI calls)

Figure 1: Benchmark Generation System - Block Diagram

The contributions of this work are (1) a demonstration and evaluation of the feasibility of automatically converting parallel applications into human-readable benchmark codes and (2) an approach for resembling the original performance by generating benchmarks from communication traces.

Our work benefits application developers, communication researchers, and HPC system designers. Application developers can benefit in multiple ways. First, they can quickly gauge the application performance of a target machine before investing in the effort to port their applications to that machine. Second, they can use the generated benchmarks for performance debugging as the benchmarks can separate communication from computation to help isolate observed performance anomalies. Third, application developers can examine the impact of alternative application implementations such as different data decompositions (causing different communication patterns) or the use of computational accelerators (reducing computation time without directly affecting communication time). Communication researchers can benefit by being able to study the impact of novel messaging techniques without the need to build complex applications and without access to source code that is not freely distributed or even classified. Finally, procurement of HPC systems can benefit by contracting vendors to deliver a specified performance on a given auto-generated benchmark without having to provide those vendors with the actual application.

[*]This work was supported in part by NSF grants 1058779, 0958311, 0937908. It used resources of the National Center for Computational Sciences at ORNL and was performed in part at ORNL, managed by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725.

PMBS'11, November 13th, 2011, Seattle, Washington, USA.
ACM 978-1-4503-1102-1/11/11.

2. BENCHMARK GENERATOR DESIGN

The process of automatic benchmark source code generation from communication traces is accomplished by traversing through the trace of a parallel application obtained from ScalaTrace. We utilize ScalaTrace [2] for communication trace collection. ScalaTrace captures the communication in lossless and near constant size with respect to the number of nodes and timesteps. 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. Power-RSDs (PRSD) recursively specify RSDs nested in a loop. It also employs pattern based intra-node and inter-node compression techniques to extract the application's communication structure.

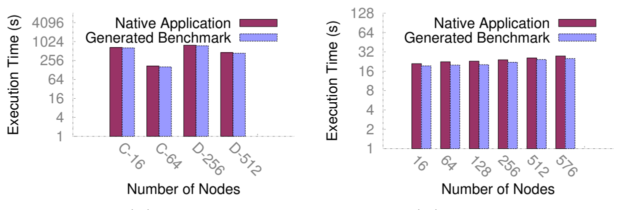
During Traversal, PRSDs representing loops are converted to C-style for loops. Behavioral constraints captured by traces are imposed in the generated code using conditionals on loop index variables and on ranks of the processes participating in a particular event. The RSDs that represent point-to-point communication are converted to respective point-to-point MPI calls in C code. For example, blocking sends and receives are transformed to MPI_Send and MPI_Recv.Collective calls are generated using MPI collective routines in C such as MPI_Barrier, MPI_Reduce, MPI_Alltoall and so on. The communicator-based MPI events are converted to the respective routines such as MPI_Comm_split and MPI_Comm_dup.

We generate delta times for the computational regions. In a compressed trace, delta times before the leading event of nested loops are represented by a list of histograms to distinguish between different execution paths. During code generation, conditionals on loop iterator variables are generated so that different execution paths will lead to different sleep times. Histograms contain the statistical information, e.g, min, max, and mean of the recorded delta times. Currently, we use the mean to generate sleep statements. For applications exhibiting heavy load imbalance, more fine-grained approaches can be used to generate sleeps.

3. EXPERIMENTAL RESULTS

To evaluate our benchmark generation tool, we generated C code with MPI calls for the NAS Parallel Benchmarks (NPB) suite (version 3.3 for MPI) using class C and D input sizes [1] and for the Sweep3D neutron-transport kernel [3].

Our first set of experiments verifies the correctness of the generated benchmarks, i.e., the benchmark generator's ability to retain the original applications' communication pattern. These experiments were done on ARC, a cluster with 1728 cores on 108 compute nodes, 32 GB memory per node and an Infiniband Interconnect. For these experiments, we acquired traces of our test suite, generated communication benchmarks, and executed these benchmarks also on the same machine. We linked both the generated codes and the original applications with mpiP. Experimental results (not presented here) showed that, for each type of MPI event, the event count and the message volume measured for each generated benchmark matched perfectly with those measured for the original application. We then instrumented each generated benchmark with ScalaTrace and compared its communication trace with that of its respective original application. The results (again, not presented here) show



(a) LU (b) Sweep3D

Figure 2: Timing Accuracy of NPB LU and Sweep3D Codes

that the original applications and the generated benchmarks have equivalent traces.

In the second experiment, we assessed the ability of the generated benchmark to retain the performance in terms of wall-clock time relative the original application. We executed both the original application and the generated benchmark on ARC, measured and compared the elapsed times. The results obtained are shown in the Figure 2. Quantitatively, the mean percentage error $(|T_{gen} - T_{app}|/T_{app} * 100\%)$ across all the graphs is only 6.7%.

4. CONCLUSION

We have designed and implemented a novel communication benchmark code generator that generates benchmark code in C with MPI calls from communication traces. These traces are generated by ScalaTrace, a lossless and scalable framework to extract communication, I/O operations and execution time while abstracting away the computations. These benchmarks are human readable, compact, easy to generate and port. They also preserve the behavior of the original application in terms of execution time, communication volume and ordering of events. Furthermore, application code is obfuscated by our benchmark generation process, which allows auto-generated benchmarks of otherwise restricted / distribution-controlled applications to be released to the public. Experiments were done with NPB and Sweep3D. The results show that the benchmarks accurately preserve not only application semantics but also overall execution time. Our benchmark generator can benefit application developers, communication researchers and HPC system designers. It may assist in performance analysis of software, hardware and can also ease migration of applications across different platforms. The full paper can be found at [4].

5. REFERENCES

[1] Bailey, D. H., *et. al* The NAS Parallel Benchmarks. *The International Journal of Supercomputer Applications 5*, 3 (Fall 1991), 63–73.

[2] Noeth, M., Mueller, F., Schulz, M., and de Supinski, B. R. Scalable compression and replay of communication traces in massively parallel environments. In *International Parallel and Distributed Processing Symposium* (Apr. 2007).

[3] Wasserman, H., Hoisie, A., and Lubeck, O. Performance and scalability analysis of teraflop-scale parallel architectures using multidimensional wavefront applications. *Int. J. High Perform. Comput. Appl. 14* (2000), 330–346.

[4] Deshpande, V., Wu, X., and Mueller, F. Auto-Generation of Communication Benchmark Traces. *SIGMETRICS Performance Evaluation Review 40(2)* (2012)