Combining Phase Identification and Statistic Modeling for Automated Parallel Benchmark Generation

Ye Jin, Mingliang Liu, Xiaosong Ma
NCSU, USA and QCRI, Qatar
yjin6@ncsu.edu
yjin,mliu,xma@qf.org.qa

Qing Liu, Jeremy Logan, Norbert Podhorszki, Jong Youl Choi, Scott Klasky
Oak Ridge National Lab, USA
liuq,loganjs,pnorbert,choij,klasky@ornl.gov

Abstract

Parallel application benchmarks are indispensable for evaluating/optimizing HPC software and hardware. However, it is very challenging and costly to obtain high-fidelity benchmarks reflecting the scale and complexity of state-of-the-art parallel applications. Hand-extracted synthetic benchmarks are time- and labor-intensive to create. Real applications themselves, while offering most accurate performance evaluation, are expensive to compile, port, reconfigure, and often plainly inaccessible due to security or ownership concerns.

This work contributes APPRIME, a novel tool for trace-based automatic parallel benchmark generation. Taking as input standard communication-I/O traces of an application's execution, it couples accurate automatic phase identification with statistical regeneration of event parameters to create compact, portable, and to some degree reconfigurable parallel application benchmarks. Experiments with four NAS Parallel Benchmarks (NPB) and three real scientific simulation codes confirm the fidelity of APPRIME benchmarks. They retain the original applications' performance characteristics, in particular the relative performance across platforms.

Categories and Subject Descriptors D.4.8 [*Performance measures*]

General Terms Performance

Keywords HPC applications, trace, phase identification, statistical profiling, automatic benchmark generation

1. Introduction

Benchmarks play a critical role in evaluating hardware and software systems. Compared to CPU, database, and mobile test workloads, supercomputing benchmarks are especially challenging and costly to construct or acquire. With both the scale (in terms of problem size and parallelism) and the complexity of applications growing alongside machine sizes, kernel-based benchmarks such as the NPB suite [8] fail to portrait state-of-the-art applications (e.g., multi-physics codes). Meanwhile, hand extracted benchmarks based on real-world, large-scale applications (such as FLASHIO [6] and GTCBench [3]) are highly labor-intensive to create and cannot easily keep up with the evolution of their long-lived base applications.

To this end, recent research developed tools for automatic generation of communication benchmarks based on trace compression and replay [9, 11]. The automatically generated codes can keep up with the original application's evolution rather easily. However, replay-based benchmarks have several intrinsic drawbacks. They require the use of a specialized, compression-enabled tracing library and cannot leverage other formated existing traces or standard tracing libraries.

There are also recent projects investigating the creation of reconfigurable benchmarks. However, such tools possess significant restrictions. The more general-purpose benchmark tools [2, 4] require users to "assemble" a synthetic benchmark from a limited number of key workload characteristics such as instruction mix and instruction-level parallelism. Skel [7], a parallel I/O benchmark creation tool, also requires users to clearly identify the begin and end points of periodic I/O phases. In addition, they are not designed to include computation or communication activities.

We propose APPRIME, an automatic benchmark generation tool based on off-line statistical trace profile extraction. Given an iterative parallel simulation (the most common type of large-scale HPC applications), APPRIME takes as input the set of traces generated by parallel processes in one execution of the original application and automatically generates as output *benchmark source code* with similar computation, communication, and I/O behavior. This result benchmark comes with a concise configuration file for users to set execution parameters, such as the number of timesteps and checkpoint frequency. Rather than only aiming at future parallel trace replay, APPRIME strives to "understand" (to some extent) an application and create a stand-alone benchmark that imitates its behavior.

Unlike trace replay-based benchmark creation, APPRIME obtains information from traces but takes a "statistical view" of applications, where a benchmark should reproduce the distributions rather than line-by-line repetition of traced events. On the other hand, it recognizes that parallel programs are usually tightly coupled codes, whose executions do not build on random events: activities across processes and across timestep iterations are highly correlated (if not identical). Therefore APPRIME takes a hybrid approach, with (1) automatic trace-based phases (timesteps) identification through string analysis, (2) Markov-Chain-based timestep behavior model to enable re- configurable execution length (number of timesteps), and (3) statistical regeneration of event parameters.

This work presents our first step, a proof-of-concept prototype focusing on *re-producing communication and I/O activities*, with computation emulated with a rather simplistic manner (by sleep intervals whose durations are generated statistically). The emulation of computation activities, planned as future work, is to be plugged in as a building block.

2. APPrime Design

We design our APPRIME prototype to validate the idea of generating accurate yet reconfigurable benchmark based on statistical summarized *profile* of traces, without retaining or replaying the

PPoPP'15, February 7–11, 2015, San Francisco, CA, USA
ACM 978-1-4503-3205-7/15/02
http://dx.doi.org/10.1145/2688500.2688541

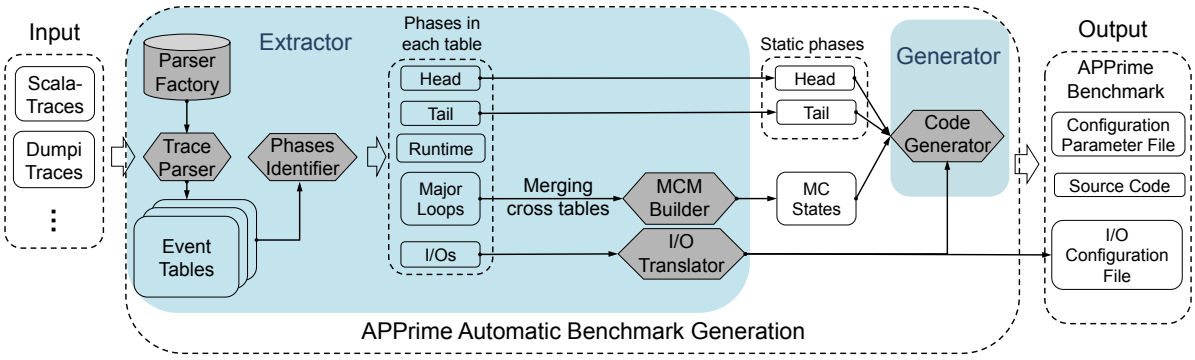
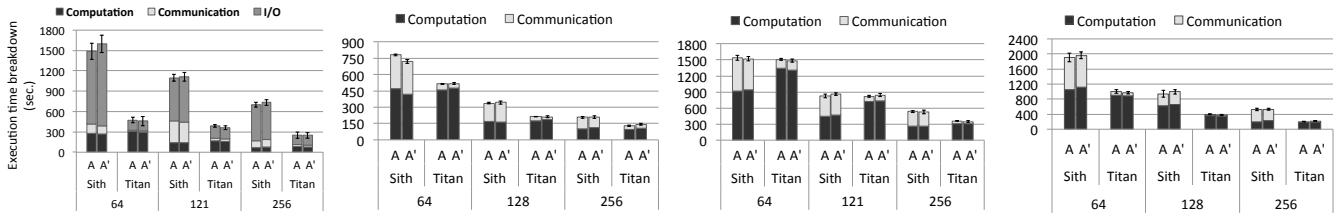


Figure 1. APPRIME overall workflow. The left part is input, right part output. It has two phases, which are trace profile extractor and benchmark generator respectively.



(a) BTIO (b) CG (c) SP (d) LU

Figure 2. Computation, communication and I/O phases time of selected NAS benchmarks on two platforms

original/decompressed trace. Figure 1 illustrates the software architecture of APPRIME and its two-phase workflow: *trace profile extraction* and *automatic configurable benchmark generation*.

To construct a parametric benchmark with APPRIME, users need to provide the following input: (1) a set of traces from one or more prior executions of the target application, (2) the number of timesteps executed in the traced run, and (3) the frequency of each type of periodic I/Os (such as checkpoint and result-snapshot output).

3. Evaluation

Our tests used two platforms of Oak Ridge Leadership Computing Facility (OLCF) cluster (Sith) and supercomputer (Titan).

We validate APPRIME's major design choices using the following use cases: Case 1 (cross-platform performance assessment), we used the four most communication-intensive members (BTIO, SP, CG, and LU) of the NAS benchmark suite [1], see results in Figure 2; case 2 (I/O configuration evaluation), we started with three large real-world applications (all proprietary), whose developers/users are interested in exploring asynchronous I/O through data staging: the quantum turbulence code BEC2 and two gyro-kinetic particle simulations: XGC and GTS. All traces are collected with the SST DUMPI library [5]. Results show that APPRIME consistently identifies correct phases in traces and effectively summarizes the phases static and dynamic characteristics into Markov Chain Models with high predictability.

4. Future Work

Our on-going work is investigating ways to regenerate synthetic computation activities simulating real-application workloads. This can be viewed as a recursive step within APPRIME, where we "zooms into" the computation intervals and statistically regenerate integer/floating-point computation instructions (along with memory access patterns) as observed from the original application. Also, the current APPRIME prototype focuses on temporal behavior study and has not yet studied creating benchmarks with similar *scalability behavior* when the number of processes is changed. We suspect that this can be done by learning an application's scaling behavior and applying techniques such as communication pattern extrapolation [10]. Finally, APPRIME might be made more efficient and flexible by enabling online trace processing (to distill patterns and statistics for benchmark creation).

References

[1] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga. The nas parallel benchmarks. In *IJSA*, 1991.

[2] J. Dujmović. Automatic Generation of Benchmark and Test Workloads. In *WOSP/SIPEW*, 2010.

[3] gtc2link. GTC-benchmark in NERSC-8 suite, 2013.

[4] A. M. Joshi, L. Eeckhout, and L. K. John. The Return of Synthetic Benchmarks. In *SPEC Benchmark Workshop*, 2008.

[5] J. P. Kenny, G. Hendry, B. Allan, and D. Zhang. Dumpi: The mpi profiler from the sst simulator suite, 2011.

[6] R. Latham, C. Daley, W. keng Liao, K. Gao, R. Ross, A. Dubey, and A. Choudhary. A case study for scientific i/o: improving the flash astrophysics code. *CSD*, 5(1):015001, 2012.

[7] J. Logan, S. Klasky, H. Abbasi, Q. Liu, G. Ostrouchov, M. Parashar, N. Podhorszki, Y. Tian, and M. Wolf. Understanding I/O Performance Using I/O Skeletal Applications. In *Euro-Par*. Springer-Verlag, 2012.

[8] NASA. Nas parallel benchmarks. http://www.nas.nasa.gov/publications/npb.html, 2003.

[9] X. Wu, V. Deshpande, and F. Mueller. ScalaBenchGen: Auto-Generation of Communication Benchmarks Traces. In *IEEE IPDPS*, 2012.

[10] X. Wu and F. Mueller. ScalaExtrap: Trace-based Communication Extrapolation for SPMD Programs. In *ACM PPoPP*, 2011.

[11] Q. Xu, J. Subhlok, R. Zheng, and S. Voss. Logicalization of Communication Traces from Parallel Execution. In *IEEE IISWC*, 2009.