perf-taint: Taint Analysis for Automatic Many-Parameter Performance Modeling

Marcin Copik
marcin.copik@inf.ethz.ch
ETH Zürich
Zürich

Torsten Hoefler (advisor)
htor@inf.ethz.ch
ETH Zürich
Zürich

ACM Reference Format:
Marcin Copik and Torsten Hoefler (advisor). 2019. perf-taint: Taint Analysis for Automatic Many-Parameter Performance Modeling. In *Proceedings of The International Conference for High Performance Computing, Networking, Storage, and Analysis (SC19).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 POSTER DESCRIPTION

The design process of a massively parallel program requires a deep understanding of computational kernels and communication patterns that are present in the application. Performance modeling has become a standard technique to solve problems such as locating scalability bottlenecks or predicting program behavior for varying problem size and on a new architecture[2]. The goal of parametric performance modeling is to estimate a model that expresses the performance of an application in terms of a purely analytical expression[5] that depends on one or more parameters. The traditional approach follows three major steps: parameter identification, designing an experiment to measure the influence of changes in parameters, and estimating the best model for the provided data, as shown on the right part of Figure 1. Since accurate models cannot be generated when focusing on a single parameter in isolation, the user is required to choose important parameters and understand how their combinations influence the program behavior. Otherwise, the user has to consider all possible combinations and allocate a significant amount of compute time for experiments. These tasks require significant user expertise in modern scientific applications. Due to the black-box nature of empirical modeling, the modeler cannot distinguish between actual runtime and effects of noise on measurements, leading to overfitting, estimating false dependencies, and generating incorrect models for constant functions with negligible execution time. Even with a significant degree of automation, performance modeling of an application remains complex and time-consuming[1].

perf-taint. We propose *perf-taint*, a hybrid program analysis integrated with Extra-P, to supply program information

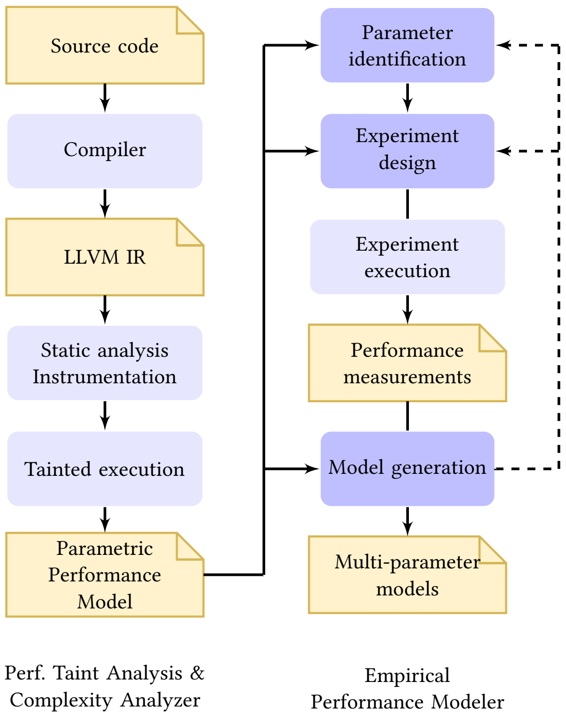

Figure 1: Automatic multi-parameter performance modeling in Extra-P[3], augmented with taint analysis.

to the modeling process and achieve the following goals: determine which parameters affect the complexity of functions, provide users with information on parameter dependencies, decrease the instrumentation overhead, and prune false and overfitted models. We use taint analysis, a computer security technique which reliably tracks propagation of input values in the program, to build a Parametric Performance Profile for each function, as shown in Figure 1.

Our analysis is partially dynamic since static techniques are often unable to provide precise answers even for seemingly simple analyses. Determining if a given configuration parameter affects certain parts of a program is undecidable, as known from Rice's theorem[6]. While Rice's theorem

SC19, November 17–22, 2019, Denver, Colorado
2019. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

shows that a complete analysis is impossible, the development of an approximate analysis for only a subset of programs is in practice difficult. The main culprits are abstraction overhead, complex abstract data types, and run-time configurability of programs. Simplifying the code relies on pointer aliasing which is a hard problem, where the most advanced inter-procedural techniques are very imprecise on large code bases[7].

Parametric Performance Profile. Taint analysis[4] marks and tracks the movement of certain data elements and their computed results through the execution of a program. For each program data location, a corresponding location in the shadow memory is used to store taint labels. The analysis is expressed with three main components: taint sources, propagation policy, and taint sinks. We expose a simple API to let users define program parameters as taint sources and implicitly taint performance-critical parameters in libraries, such as the number of parallel workers in OpenMP and MPI. There are two ways to propagate labels between memory locations: data-flow, passing labels from inputs of operations to their outputs, and control-flow based where the propagation happens through a control dependence. For each function, we instrument control-flow instructions affecting performance with taint sinks. Thus, we can inspect loop exit conditions to determine if its iteration count is affected by parameters. Results are aggregated across nested functions calls and generated for each function invocation, distinguished by the callpath. We build a Parametric Performance Profile, a model that allows understanding which parameters affect the computation volume and identifying multiplicative dependencies of parameters either through nested loops or the presence of multiple labels.

Results. We evaluate our hybrid modeling framework on two benchmarks: `LULESH` and `su3_rmd` from MILC suite. In `LULESH`, we look for three parameters and from 347 functions, we prune 294 statically and 9 dynamically. In `su3_rmd`, we look for four parameters and prune 364 out of 621 functions. By instrumenting only relevant functions, we decrease the overhead of experiments up to 45 times in LULESH. We observe higher improvements in C++ applications due to object-oriented and template abstractions. We apply the parametric profile to the model estimation and help Extra-P to avoid incorrect models. We generally observe that new models are closer to (nearly always often exactly matching) the ground truth established with manual performance analysis [5, Section 4].

Our work is first to introduce taint analysis as a powerful concept for performance modeling and demonstrate that good performance modeling must combine properties from the program analysis with empirical performance modeling and that a hybrid program analysis can overcome the inherent limitations of static techniques. As a result, performance modeling is now possible for large scientific applications with many performance parameters and little manual preparation.

REFERENCES

[1] Alexandru Calotoiu, David Beckingsale, Christopher W. Earl, Torsten Hoefler, Ian Karlin, Martin Schulz, and Felix Wolf. 2016. Fast Multi-Parameter Performance Modeling. In *Proc. of the 2016 IEEE International Conference on Cluster Computing (CLUSTER), Taipei, Taiwan.* IEEE Computer Society, 172–181.

[2] Alexandru Calotoiu, Alexander Graf, Torsten Hoefler, Daniel Lorenz, Sebastian Rinke, and Felix Wolf. 2018. Lightweight Requirements Engineering for Exascale Co-design. IEEE. IEEE International Conference on Cluster Computing (Cluster'18).

[3] A. Calotoiu, T. Hoefler, M. Poke, and F. Wolf. 2013. Using Automated Performance Modeling to Find Scalability Bugs in Complex Codes. IEEE/ACM International Conference on High Performance Computing, Networking, Storage and Analysis (SC13).

[4] James Clause, Wanchun Li, and Alessandro Orso. 2007. Dytan: a generic dynamic taint analysis framework. In *Proceedings of the 2007 international symposium on Software testing and analysis.* ACM, 196–206.

[5] Torsten Hoefler, William Gropp, William Kramer, and Marc Snir. 2011. Performance Modeling for Systematic Performance Tuning. In *State of the Practice Reports (SC '11).* ACM, New York, NY, USA, Article 6, 12 pages.

[6] Henry Gordon Rice. 1953. Classes of recursively enumerable sets and their decision problems. *Trans. Amer. Math. Soc.* 74, 2 (1953), 358–366.

[7] Marc Shapiro and Susan Horwitz. 1997. The effects of the precision of pointer analysis. In *International Static Analysis Symposium.* Springer, 16–34.