Automatic Performance Modeling of Configurable Scientific Software

Larissa Schmid
larissa.schmid@kit.edu
Karlsruhe Institute of Technology

Abstract

Modern software is configurable and allows users to set many parameters according to their needs. Due to many non-functional parameters, usually, the same functionality can be achieved with varying performance. Performance models express application performance as functions of input parameters, helping users and developers understand application behavior. Automatic performance modeling approaches can generate performance models automatically from empirical measurements of the software. Current modeling approaches employ heuristics for deciding which configurations to measure, resulting in a trade-off between the cost of measurements and accuracy of the model. To overcome this trade-off, we propose approaches to derive the smallest necessary measurement setup based on results of a system analysis, and to automatically identify performance-irrelevant options. Our evaluation with real-world applications show that we can significantly decrease cost of performance modeling while maintaining accuracy of the resulting models.

1 Introduction

Scientific software is crucial in various fields, such as biology, physics, and material science. Scientific simulations and data analyses often require processing large amounts of data and performing complex calculations, which can take a lot of time. Hence, performance is a critical quality attribute for scientific software as it affects how quickly and how much research can be done. Usually, it is possible to achieve the same functionality with varying performance due to the many configuration options offered by the software. However, it is unclear how these configuration options affect performance, for example, when trying to understand application scalability – especially since there can be hundreds of different options to set.

Performance models express application performance as functions of input parameters, helping users and developers understand application behavior. While performance models can be created manually, the cost associated with this is high due to the need for performance experts. Therefore, developers often select an arbitrary subset of the configuration space for evaluation. Automatic performance modeling generates models from empirical measurements that cover all configuration options. However, the number of options makes exhaustive measurements of all configurations infeasible. For applications with many parameters, modeling all options results in a trade-off between model quality and the number of measurements, leading to modeling only a subset of options again or imprecise models.

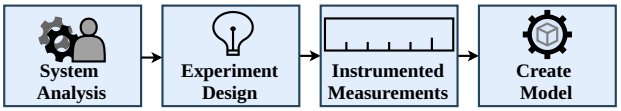

Figure 1: Automatic Performance Modeling Process.

Current approaches do not consider leveraging known interactions between parameters for designing experiments or pruning the parameter space before the modeling process. To address this gap, we present two contributions: Performance-Detective [5], a white-box measurement methodology for deducing experiment designs, and an approach to identify performance-irrelevant parameters automatically [6].

2 Automatic Performance Modeling

Constructing empirical performance prediction models automatically usually involves four phases (see Figure 1): First, the user selects parameters to consider for the model and the software system under consideration is analyzed. Second, the configurations to execute are determined in the experiment design phase. This involves either creating a full-factorial experiment design with five values per parameter [1] or utilizing a sampling strategy to sample configurations to measure from all possible configurations. Sampling strategies can, for example, rely on achieving a specific coverage, mathematical criteria, or sample configurations randomly. Next, the application is executed with the configurations derived from the experiment design phase to collect the required sample measurements. This step is the most expensive as the user needs to conduct the performance measurements on the system they want to have a performance model for, typically a high-performance system. In the last step, the user supplies the measurements to the performance modeling tool that creates the empirical performance model using a machine learning approach, such as Classification and Regression Trees or Multiple Linear Regression.

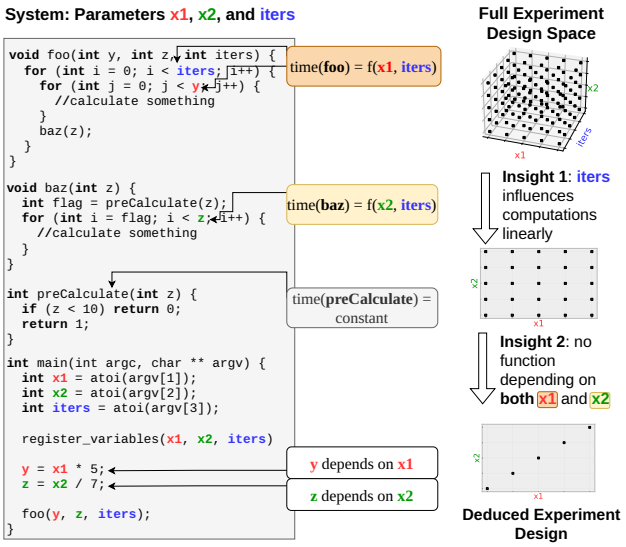

Figure 2: Deduction process of Performance-Detective.

3 Performance-Detective

Current modeling approaches do not propose a deterministic methodology for users to derive minimal experiment designs for applications with many parameters and numerical configuration options. They also require repeated measurement execution to account for measurement noise. With Performance-Detective, we use insights about the interaction of configuration options to derive a minimal experiment design.

Approach. Figure 2 illustrates our approach using an example application that takes the configuration options `x1`, `x2`, and `iters` as input and performs calculations based on them. First, we analyze the system using Perf-Taint [3], mapping the influence of configuration options to the application's functions. We consider a function's performance dependent on a configuration option if the option influences the iteration count of a loop inside the function. We show the dependencies in the example in the colored boxes: `foo` depends on `x1` and `iters`, `bar` on `x2` and `iters`, and the runtime of `preCalculate` is constant.

Using this information, Performance-Detective can now deduce a minimal experiment design. From a black-box view of the system, we start with a full-factorial experiment design that has to consider all three configuration options. First, Performance-Detective detects that `iters` influences the computations linearly. This means that measuring different values of `iters` will not provide additional insight, and we can thus strike out these configurations from the experiment design. While the linear influence is clearly visible in our example, we can verify this in real applications by measuring the loop iteration times and checking that the coefficient of variation between them is sufficiently small. Second, we see that no function is influenced by both `x1` and `x2`, meaning that these configuration options do not interact. Thus, varying

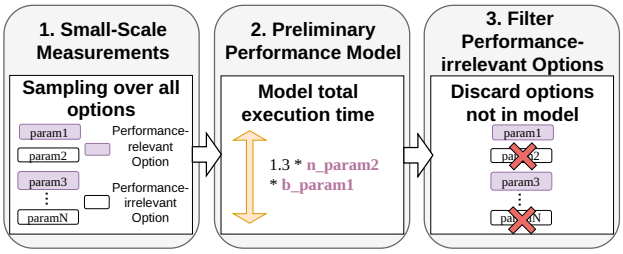

Figure 3: Process of identifying performance-irrelevant configuration options.

them independently from one another will not provide us further insight, and we can strike out these measurements. This effectively reduces the dimensionality of the experiment. In our example, we can deterministically reduce the number of experiments from 125 to only five without compromising accuracy.

Next, we execute the experiments using instrumentation-based profiling to capture the time spent in each function in one program execution. To reduce measurement overhead, we only instrument functions identified as dependent on one of our configuration options. Current modeling approaches require five repetitions of experiments to account for measurement noise. However, if Performance-Detective detects a linear influence of a configuration option on the main computation, we can skip the repetitions and could even stop the measurement after collecting data for five executions of the main computation loop.

Evaluation. Performance-Detective is orthogonal to the learning method used for creating models from measurements. We evaluate against two state-of-the-art modeling workflows by Calotoiu et al. [1], requiring a full-factorial experiment design, and Weber et al. [4], requiring a Plackett-Burman design. We use two case studies: Pressure calculation in the multi-physics framework Pace3D (4 options) and Kripke, a particle transport proxy application (3 options). We evaluate the accuracy of the models by comparing their predictions against inter- and extrapolated evaluation measurements. For both case studies, we can maintain the accuracy of the model generated with our experiment design deduced by Performance-Detective (25 and 5 experiment executions) as compared to a full-factorial design (625 and 125 experiment executions) and Plackett-Burman sampling (245 and 50 experiment executions). Across evaluation scenarios, Performance- Detective significantly decreases the cost of experiments while not meaningfully impacting model quality.

4 Reducing Parameter Space

While configuration spaces of scientific software are large, usually, not all configuration options of an application are performance-relevant. Therefore, the challenge of selecting which parameters to include in the

performance modeling process arises. While for some parameters, such as the problem size, it is usually clear that they are performance-relevant, others are more difficult to categorize: The impact of each option can vary significantly depending on the scenario computed, and the interplay between options can create complex dependencies. Tools for system analysis, such as Perf-Taint [3], as presented in Section 3, extract performance influences of options based on static and dynamic analysis automatically but cannot quantify their influence. This means that the experiment design has to consider each option, even if their performance impact is minimal.

Approach. To guide domain scientists with parameter selection for performance modeling, we present a novel approach to ease the process. Our approach introduces a pre-processing step that automatically determines performance-irrelevant configuration options and removes them from the further modeling process. Figure 3 shows an overview of our approach. First, we gather samples by conducting measurements that consider all configuration options of the application. To keep the cost low, we execute them on a small scale, considering only two different values for numeric options (high/low) and using small yet realistic problem sizes. Second, we build a preliminary performance model from the collected samples by re-using an existing modeling method. Based on this performance model, we can classify all options as either performance-relevant if they are used within the model to make predictions or performance-irrelevant if they are not used. These performance-irrelevant configuration options can safely be disregarded for the further modeling process.

Evaluation. We use DECART [2] as an exemplary modeling method and Pace3D as a case study to evaluate our approach. We consider a scenario in Pace3D that offers 34 non-functional configuration options. As DECART supports only binary configuration options, we map every non-binary configuration option to a binary representation, resulting in 66 binary options. Moreover, DECART uses random sampling, recommending as many samples as there are options to start with and then iteratively expanding the number of samples depending on the quality of the resulting model. We, therefore, generate ten models with 66 to 660 samples.

First, we measure the relative runtime deviation when changing options identified as irrelevant and interview one of Pace3D's main developers to get their assessment on the performance relevance of options. Our evaluation shows that our approach can accurately identify performance-irrelevant options while not misjudging performance-relevant options. Second, we evaluate the cost savings for creating performance models by employing our approach for black-box and white-box workflows. While we cannot quantify the cost saved for black-box workflows, we expect the same number of samples to result in better model quality, as samples that do not provide insight into system performance are eliminated by pruning performance-irrelevant options. For white-box workflows, we show that the saved cost during the performance modeling process will outweigh the additional cost incurred by our pre-processing step for white-box workflows.

5 Conclusion

We propose two approaches for different stages of the performance modeling process to reduce the cost of automatic performance modeling approaches while maintaining the accuracy of the resulting performance models: First, we introduce an approach to automatically determine performance-irrelevant configuration options, allowing for their exclusion from the performance modeling process. Our evaluation shows that we can accurately identify performance-irrelevant options, while the potential savings from applying our approach are expected to outweigh its additional costs. Second, Performance-Detective can reduce the dimensionality of the experiment design based on insights into parameter interaction and main loops. Using two case studies for the evaluation, we show that this results in significant savings regarding the number of experiments and core hours needed.

References

[1] A. Calotoiu et al. "Fast Multi-parameter Performance Modeling". In: *2016 IEEE International Conference on Cluster Computing (CLUSTER)*. Taipei, Taiwan: IEEE, Sept. 2016.

[2] J. Guo et al. "Data-Efficient Performance Learning for Configurable Systems". In: *Empirical Softw. Engg.* 23.3 (June 2018), pp. 1826–1867.

[3] M. Copik et al. "Extracting Clean Performance Models from Tainted Programs". In: *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. PPoPP '21. Virtual Event, 2021.

[4] M. Weber et al. "White-Box Performance-Influence Models: A Profiling and Learning Approach". In: *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. Madrid, Spain: IEEE, May 2021.

[5] L. Schmid et al. "Performance-Detective: Automatic Deduction of Cheap and Accurate Performance Models". In: *Proceedings of the 36th ACM International Conference on Supercomputing*. ICS '22. Virtual Event, 2022.

[6] L. Schmid et al. "Cost-Efficient Construction of Performance Models". In: 4th Workshop on Performance EngineeRing, Modelling, Analysis, and VisualizatiOn STrategy (PERMAVOST '24). Pisa, Italy, 2024.