

Practical Empirical Performance Modeling for CFD Applications Using Extra-P

Lukas Rothenberger^{a,*}, Gustavo de Morais^a, Alexander Geiß^a and Felix Wolf^a

^aTechnical University of Darmstadt, Department of Computer Science, Laboratory for Parallel Programming, Hochschulstr. 10, 64289 Darmstadt, Germany

ARTICLE INFO[†]

Keywords:
Empirical Performance Modeling;
Instrumentation Filtering;
OpenFOAM

ABSTRACT

Performance models facilitate the development and optimization of scalable applications for HPC systems. Automatic empirical performance modeling allows the creation of performance models for CFD applications and other large software suites, although challenges regarding profiling time and model accuracy arise from their size and characteristics. To mitigate these challenges, we propose an automatic tool for Score-P filter file creation. We measure exemplary OpenFOAM CFD applications using these filter files and employ Extra-P to generate strong-scaling performance models and identify scalability bottlenecks.

1. Introduction and motivation

Understanding performance at scale and identifying potential bottlenecks are crucial for developing and optimizing efficient HPC applications. While computation- and communication-intensive kernels/functions are typically well understood, implicit performance bottlenecks, such as those arising from caching or synchronization effects, can be easily overlooked. These aspects typically only manifest as significant issues once the scale of the utilized HPC system reaches a certain magnitude. To improve the utilization of computational resources and consequently reduce energy consumption, it is important to identify such bottlenecks as early as possible in the development or optimization process, utilizing a minimal amount of computational resources. This is particularly critical when dealing with large, highly configurable code bases, such as modern CFD solvers. Mathematical performance models are suitable for examining scaling behavior and identifying potential bottlenecks. However, designing these models analytically for an entire large code base is often impractical due to the manual effort required. On the other hand, automatic empirical performance modeling offers a solution to circumvent the challenges of analytical modeling.

For instance, Extra-P [1] is a state-of-the-art performance modeling tool that leverages profiling results from program executions with relatively small yet representative input data. It generates performance models for entire code bases or instrumented sections without requiring significant manual effort aside from code instrumentation. However, particularly in the case of CFD applications, the runtime and memory overhead introduced by profiling and model creation can become significant due to the sizes of modern software suites. Filtering the profiled code regions can mitigate this problem, but the definition of suitable filters might be seen as a challenge on its own. This observation led to the development of a fast, automatic, and easy-to-use tool for creating Score-P [2] filter files using a single profile. With the overarching goal of reducing the resource consumption in HPC, it is the purpose of this work to motivate and introduce the use of empirical performance models for code development and optimization by example of the open source CFD software OpenFOAM v2312 [3], the performance-modeling tool Extra-P, and the Score-P profiling framework.

2. Empirical performance modeling with Extra-P

Extra-P is a tool to automatically generate performance models for HPC applications based on small-scale measurements that offer insights similar to those of analytical performance models. It considers a set of execution parameters, such as the number of processors or input sizes. Using measurement tools like Score-P, the empirical data is gathered for various combinations of the execution parameters. Subsequently, this data is modeled with Extra-P, which returns models based on the Performance Model Normal

[†]This paper is part of the ParCFD 2024 Proceedings. A recording of the presentation is available on YouTube. The DOI of this document is 10.34734/FZJ-2025-02482 and of the Proceedings 10.34734/FZJ-2025-02175.

*Corresponding author

✉ lukas.rothenberger@tu-darmstadt.de (L. Rothenberger);
gustavo.morais@tu-darmstadt.de (G. de Morais);
alexander.geiss1@tu-darmstadt.de (A. Geiß); felix.wolf@tu-darmstadt.de (F. Wolf)

ORCID(s): 0009-0005-8988-5427 (L. Rothenberger);
0000-0002-2139-1157 (G. de Morais); 0000-0003-4565-422X (A. Geiß);
0000-0001-6595-3599 (F. Wolf)

Form (PMNF). The PMNF is a general structure of performance models, which describes the impact of the execution parameters on a performance metric (e.g., time) as a human-readable equation using a combination of monomial and logarithmic terms. These models help unveil performance bottlenecks as the application scales.

When using Extra-P for scaling analysis of applications, one has to be aware that it is designed with a focus on weak scaling. That means Extra-P models the runtime for a single average process, given the number of processes and a problem size scaled with the number of processes. Consequently, we cannot directly model the runtime in a strong-scaling scenario, as this would have the same constant problem size for each number of processes. Instead, we can model the overall effort in the form of the sum of all processes' runtimes; this metric behaves similarly to the runtime of a single average process in the weak-scaling scenario so that it can be modeled using Extra-P.

3. Automatic generation of Score-P filter files

Score-P instruments code during compilation to measure the execution time of functions, but these measurements also include the overhead associated with the instrumentation. This overhead can disturb the performance modeling, especially for frequently accessed short-running functions/regions. A solution to reduce the instrumentation overhead is to filter the functions or kernels that necessitate performance measurements (i.e., those responsible for a substantial fraction of the total runtime). This improves the model accuracy while also reducing the time and memory required to generate the performance models. Since manually creating filter files is mostly impractical for large applications, we have developed a fast, user-friendly, and freely available tool to automatically generate Score-P filter files based on one existing Score-P profile.

3.1. Filter generation

Our design for the automatic Score-P filter generator¹ is inspired by prior research on hotspot detection [4]. Our approach examines all call paths in a single measurement to identify functions essential for modeling and those that may disrupt the modeling process. Most relevant for modeling are functions that contribute significantly to the total runtime. Conversely, functions that are frequently visited but have short runtimes potentially disrupt the modeling process. From these observations, we developed the following procedure for filtering call paths: call paths with a runtime per visit in the top 25% are always included. This criterion ensures that call paths contributing significantly to the runtime are retained. To address the potential exclusion of call paths

where individual calls have a short runtime but collectively are in the top 25% of runtimes, we include the first parent call-tree node with the number of visits below the median. Once we have identified these two sets of included call paths, we also include all prefixes of the call paths to create a more comprehensive and meaningful call tree.

3.2. Effects of filtering on OpenFOAM

To evaluate the effects of automatically generated filter files in mitigating the impact of overhead on the profiling time, we have conducted the measurements described in the following on the basis of the Cavity3D-1M² benchmark contained in the OpenFOAM HPC Benchmark Suite³. Four execution configurations were considered: (1) without

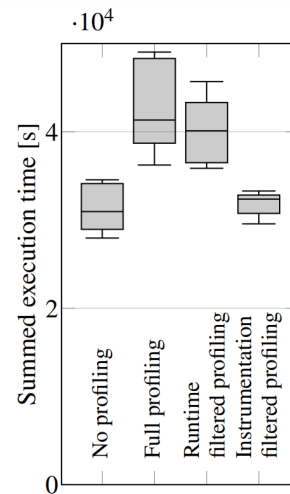


Figure 1: Effects of different filtering schemes on the required resources.

source code profiling; (2) entailing full instrumentation of the source code; (3) instrumentation filtering during runtime; and (4) filtering at the time of instrumentation. Each experiment was conducted using 520 processors distributed across five nodes of the high-performance computer Lichtenberg 2 and repeated five times⁴. The measured times do not include the time required for preprocessing. Figure 1 illustrates the summed execution times across all used processors. While the full instrumentation led to an overall increase in computation time of roughly 1.34 times in median values, it decreased to 1.30 times and 1.05 times when using the automatically generated filter file during runtime and instrumentation, respectively. Additionally, we noticed significant reductions in memory overhead and execution

²2024-05-14: <https://develop.openfoam.com/committees/hpc/-/tree/develop/incompressible/icoFoam/cavity3D>

³2024-05-14: <https://develop.openfoam.com/committees/hpc/-/tree/develop>

⁴2024-06-03: <https://www.hrz.tu-darmstadt.de/hlr/hochleistungsrechnen/index.de.jsp>

¹https://github.com/extra-p/extrap/blob/master/tools/scorep_filter_generator.py

time required for performance model creation in case a filter file was used. During development, we observed similar reductions in profiling time for other data sets, configurations, and applications.

4. Performance modeling results

We gathered performance measurements for the same 3D lid-driven cavity flow example as above on the Lichtenberg 2 cluster at TU Darmstadt. We performed a strong-scaling analysis using a mesh size of 460^3 , a limit of 3000 iterations, and 1, 2, 3, 4, 5, and 6 nodes with 104 processes per node. As mentioned, we cannot directly model the runtime for strong scaling; instead, we model the effort in core-seconds.

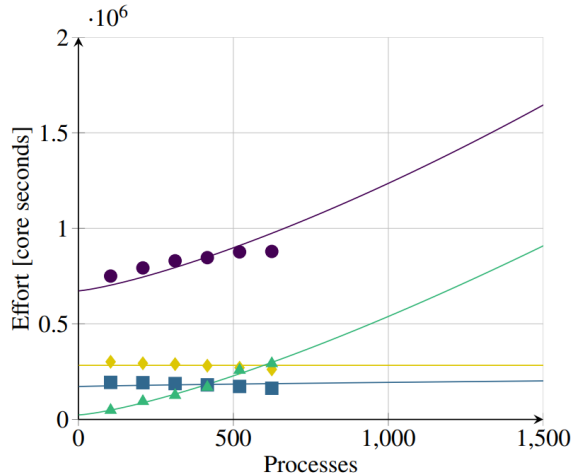


Figure 2: Effort performance models for the lid-driven cavity flow, created using measurements for 1-6 nodes.

Using this metric, perfect scaling would be represented by a constant model, meaning that the work is shared equally among all processes without any overhead. We see this in Fig. 2 for the precondition kernel, which dominates the performance of the cavity example up to 600 processes. The matrix multiplication `Amul` also contributes significantly. Beyond 600 processes, `MPI_Allreduce` of the `gSumProd` function overtakes all other kernels and dominates the performance. This also coincides with the results reported by Brogi et al. [5], who found that MPI dominates the performance when the number of cells per process gets smaller. Unlike the other functions, the `main` function’s model is created by combining the individual models of all functions within its call path. As a result, the model for the `main` function deviates from the measurement at 624 nodes. Despite this deviation, the model accurately reflects the performance trend, displaying superlinear growth similar to the model and measurements for `MPI_Allreduce`, which exhibit the same behavior. In order

to assess the quality of the created models, we extend the set of measurements by 8, 10, 12, and 14 nodes. The corresponding models are shown in Fig. 3. When comparing both figures, it is evident that Extra-P is able to identify the correct scaling behavior within a reasonable projection distance.

5. Conclusion

Performance modeling helps identify potential bottlenecks when scaling up. Unfortunately, crafting such models by hand requires expertise and is laborious for larger applications such as CFD solvers. An empirical modeling tool like Extra-P can simplify this task but requires measuring the application. In many measurements, short-running, often called functions, exhibit comparatively high run-to-run variations, which disturb the modeling process. Our filter generator automatically produces a filter file that prevents the instrumentation of small functions, thus reducing measurement overhead and improving the models. We used this approach for the Cavity3D benchmark of OpenFOAM and found that the `MPI_Allreduce` functions of the PDC solver are the main bottleneck.

Acknowledgements

The authors gratefully acknowledge the German Federal Ministry of Education and Research (BMBF) and the Hessian Ministry of Science and Research, Art and Culture (HMWK) for supporting this work as part of the NHR funding. Furthermore, the authors appreciate the computing time provided to them on the high-performance computer Lichtenberg at the NHR Center NHR4CES@TUDa. This is funded by the BMBF and the HMWK. The computations for this research were performed using computing resources under Project 2306.

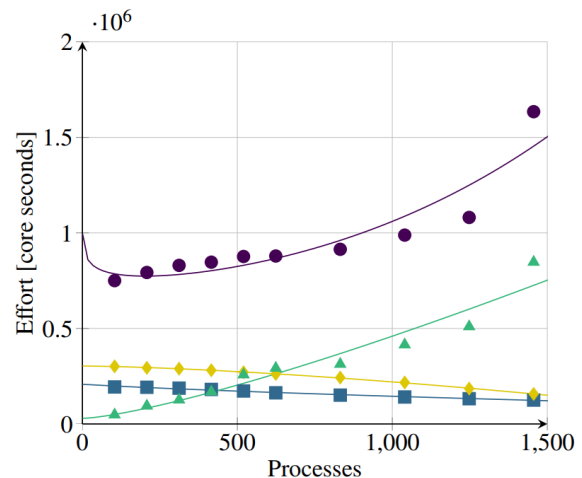


Figure 3: Effort performance models for the lid-driven cavity flow, created using an extended set of measurements.

References

- [1] A. Calotoiu, T. Hoeffler, M. Poke, F. Wolf, Using automated performance modeling to find scalability bugs in complex codes, in: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, ACM, 2013, pp. 1–12. doi:10.1145/2503210.2503277.
- [2] A. Knüpfer, C. Rössel, D. an Mey, S. Biersdorff, K. Diethelm, D. Eschweiler, M. Geimer, M. Gerndt, D. Lorenz, A. Malony, W. E. Nagel, Y. Oleynik, P. Philippen, P. Saviankou, D. Schmidl, S. Shende, R. Tschüter, M. Wagner, B. Wesarg, F. Wolf, Score-P: A joint performance measurement run-time infrastructure for Periscope, Scalasca, TAU, and Vampir, in: Tools for high performance computing 2011, Springer, 2012, pp. 79–91. doi:10.1007/978-3-642-31476-6_7.
- [3] H. Jasak, A. Jemcov, Z. Tukovic, et al., OpenFOAM: A C++ library for complex physics simulations, in: International workshop on coupled methods in numerical dynamics, Vol. 1000, 2007, pp. 1–20.
- [4] S. A. Mohammadi, L. Rothenberger, G. de Morais, B. N. Görlich, E. Lille, H. Rüthers, F. Wolf, Filtering and ranking of code regions for parallelization via hotspot detection and OpenMP overhead analysis, in: Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis, SC-W '23, ACM, New York, NY, USA, 2023, p. 1368–1379. doi:10.1145/3624062.3624206.
- [5] F. Brogi, S. Bnà, G. Boga, G. Amati, T. Esposti Ongaro, M. Cerminara, On floating point precision in computational fluid dynamics using OpenFOAM, Future Generation Computer Systems 152 (2024) 1–16. doi:10.1016/j.future.2023.10.006.