Analysis and Visualization of Important Performance Counters To Enhance Interpretability of Autotuner Output

Mohammad Zaeed[1], Tanzima Z. Islam[1], Younghyun Cho[3], Xiaoye Sherry Li[2], Hengrui Luo[2], Yang Liu[2]
[1]Texas State University, [2]Lawrence Berkeley National Laboratory, [3]University of California, Berkeley

***Abstract*— Autotuning is a widely used method for guiding developers of large-scale applications to achieve high performance. However, autotuners typically employ black-box optimizations to recommend parameter settings at the cost of users missing the opportunity to identify performance bottlenecks. Performance analysis fills that gap and identifies problems and optimization opportunities that can result in better runtime and utilization of hardware resources. This work combines the best of the both worlds by integrating a systematic performance analysis and visualization approach into a publicly available autotuning framework, GPTune, to suggest users which configuration parameters are important to tune, to what value, and how tuning the parameters affect hardware-application interactions. Our experiments demonstrate that a subset of the task parameters impact the execution time of the Hypre application; the memory traffic and page faults cause performance problems in the Plasma-DGEMM routine on Cori-Haswell.**

I. Introduction and Background

GPTune [1] is an autotuning framework that tunes high-performance application codes with the help of Bayesian optimization methodologies. However, like most autotuning frameworks, GPTune lacks good support for intuitively presenting how the tuning parameters affect hardware usage, thus missing the opportunity to provide insights to users about *how they can improve the performance of their application* while tuning their parameters.

Performance characterization fills this gap by identifying "important" interactions between the application and the underlying system, captured by hardware performance counters, that can predict the execution time of that application across many configurations. Here, a configuration can be defined as using a specific setup, e.g., input size, algorithm, and the number of tasks. However, modern computing systems provide hundreds of performance counters to explain such interactions. As a result, manually analyzing a large volume of information to uncover the impact of tuning parameters on hardware usage is computationally intractable. Our previous work provides an open-source offline-analysis-based framework, DASHING [2], for systematically identifying important interactions in predicting the performance of applications, thus filling the gap an autotuner leaves.

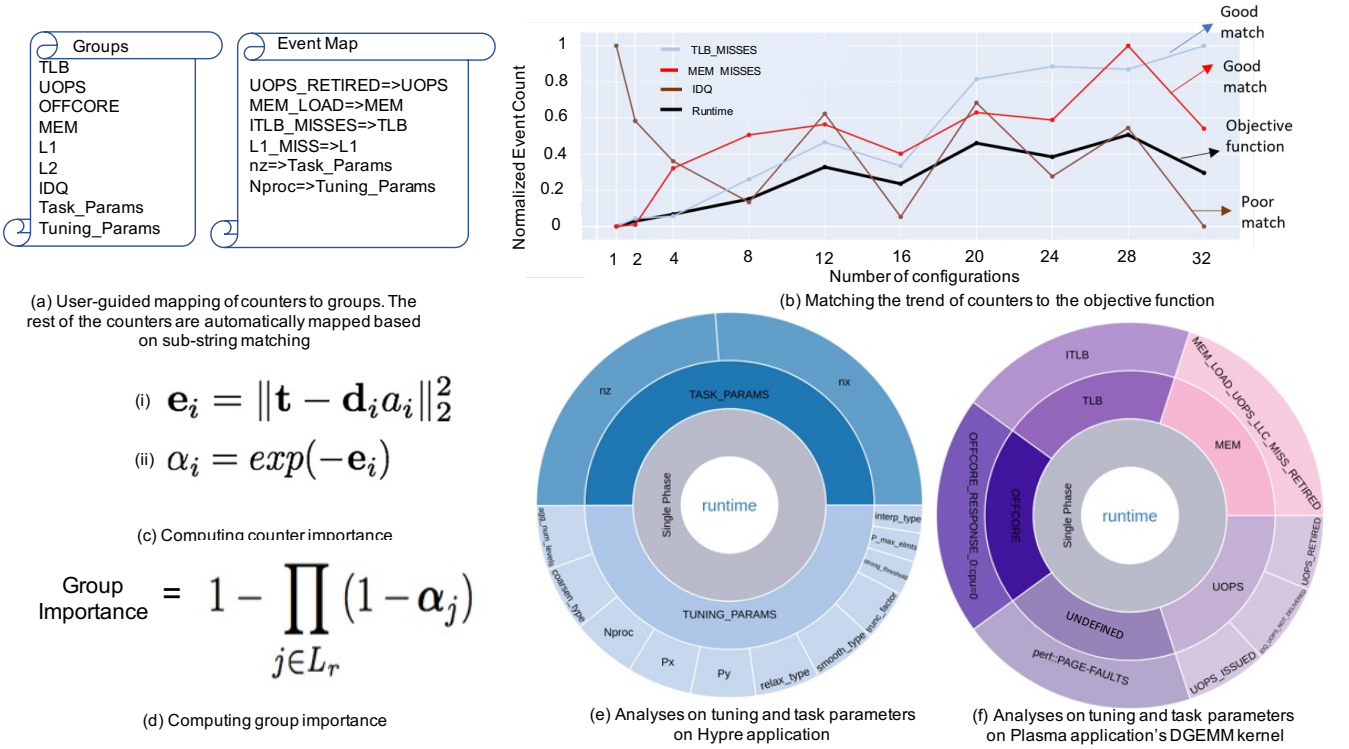
Hence, in this work, we combine the power of performance analysis with that of online autotuning by integrating DASHING with GPTune. This integration enables users to identify performance problems while achieving the original tuning goals. Users can leverage these insights to optimize application performance.

II. Challenges and Our Approach

The first technical challenge in integrating hardware performance counter-based analysis into an autotuner is that there can be hundreds of such counters to analyze. To address this challenge, DASHING maps the hardware performance counters to a few groups and then presents which groups are important to optimize. However, the mapping between performance counters and groups in DASHING relies upon the PAPI [3] tool collecting the performance counters. However, GPTune allows users to collect performance counter traces from various systems using any tool. Hence, in this work, we first extend DASHING's abilities to categorize hardware performance counters into arbitrary user-defined groups. However, automatically mapping the counters to groups for all systems is not viable due to the significant difference in the names of the performance counters. Hence, we propose a user-guided semi-automated approach to map counters to different user-defined groups.

The semi-automated process uses a sub-string matching operation to map the performance counters to groups, as described by users (Figure 1a). Users can also create a manual mapping between counters and group names (the right figure in Figure 1a). Finally, the counters that cannot be mapped to any of the groups are categorized under an "Undefined" group. Users have the opportunity to refine the mappings at a later time based on domain knowledge.

Figure 1b illustrates the rationale behind calculating the importance of a counter or parameter. As the number of configurations changes, the trend of how the red counter evolves (marked as "Good match") matches with that of the objective function (bold black line). This observation depicts that using the red counter, one can predict the shape of the objective function; hence it is "important". Mathematically, Figure 1c shows how the importance of a counter or parameter is calculated. Figure 1c-(i) calculates the prediction error if counter i is used to predict the objective function t from a dictionary of performance counters d of size MxN, where M is the number of configurations and N is the number of counters. Here, d_i corresponds to the i^{th} counter's values across all configurations and a_i corresponds to a coefficient. Thus, we formulate the problem of computing importance for each counter as a linear combination of a number of the counters. We use the ensemble Orthogonal Matching Pursuit algorithm in DASHING to calculate a_i. We then convert the prediction error of each counter e_i to a corresponding inverse quantity called belief, α_i, using the equation shown in Figure 1c-(ii). The inverse relationship



(a) User-guided mapping of counters to groups. The rest of the counters are automatically mapped based on sub-string matching

(b) Matching the trend of counters to the objective function

(c) Computing counter importance

(d) Computing group importance

(e) Analyses on tuning and task parameters on Hypre application

(f) Analyses on tuning and task parameters on Plasma application's DGEMM kernel

Fig. 1: An overview of this research and results.

ensures that a counter causing a smaller prediction error gets a higher belief/importance. Using a negative exponent creates a larger discrepancy between two similar error terms.

Finally, Figure 1d shows how DASHING combines the belief of each counter per group r, where L_r denotes the counters in group r. Here, $1-\alpha_j$ indicates the amount of information that cannot be covered by counter j, $\Pi_r(1-\alpha_j)$ denotes the total amount of information that the entire group of counters cannot cover. Hence, $1-\Pi_r(1-\alpha_j)$ calculates the aggregated information that the entire group of counters cover.

III. PRELIMINARY RESULTS

We present the performance characteristics of applications in a hierarchical manner (Figures 1e and f) by showing the importance of different phases of an application (innermost layer), important groups per phase (the next outward layer), and the important performance counters per group (the outermost layer). Since both of the applications analyzed in this work only contain a single kernel (or phase), Figures 1e and f show a single layer. For applications with multiple phases, the innermost circle shows them sorted based on their runtimes (shown in the poster). Figure 1e presents the analysis of tuning and task parameters of the Hypre application [4] for solving the Poisson equation on a structured 3D grid using the Boomer-AMG preconditioner. The application has 3 task parameters (nx, ny, nz defining the grid size in each dimension) and 12 tuning parameters, including choice and parameters of the process grid, coarsening algorithms, smoothers, and interpolation operators. We collected 410 function evaluation samples using 32 NERSC Cori [5] Haswell nodes, archived at the GPTune web (https://gptune.lbl.gov/repo/dashboard/). We analyze the importance of each task and tuning parameter against the total runtime of Hypre. Our analysis shows that as nx and nz change, the execution time of Hypre varies similarly, while the change in ny does not correlate well.

Figure 1f presents the analysis on performance counter data of the DGEMM kernel in the Plasma application [6]. We ran this experiment on a single node of Cori-Haswell with 136 function evaluations (driven by a grid-based search). The performance counter data was collected at the end of every run and represents the total number of occurrences throughout the runtime. From Figure 1f, we can observe that among hundreds of counters, the last level cache miss generating memory traffic (MEM_LOAD_UOPS_LLC_MISS_RETIRED) explains the execution time of this routine. Additionally, the number of page faults (UNDEFINED group) predicts how runtime changes across numerous configurations. Optimizations such as changing the data structure, and deleting temporary variables can reduce the number of page faults, thus improve the execution time of Plasma's DGEMM kernel.

IV. CONCLUSIONS

This work incorporates the strength of autotuning with that of performance characterization to achieve a combined goal of both identifying performance optimization opportunities and guiding users to tune "important" configurable parameters. We incorporate a principled importance analysis approach in a publicly available autotuning framework GPTune and demonstrate the applicability of our work by uncovering potential optimization opportunities. Our analysis and visualization methodologies can explain "how" applications use an underlying system—an insight that can help users design optimizations to reduce bottleneck-causing interactions.

REFERENCES

[1] Y. Liu, W. M. Sid-Lakhdar, O. Marques, X. Zhu, C. Meng, J. W. Demmel, and X. S. Li, "Gptune: Multitask learning for autotuning exascale applications," in *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 234–246. [Online]. Available: https://doi.org/10.1145/3437801.3441621

[2] T. Islam, A. Ayala, Q. Jensen, and K. Ibrahim, "Toward a programmable analysis and visualization framework for interactive performance analytics," in *2019 IEEE/ACM International Workshop on Programming and Performance Visualization Tools (ProTools)*. IEEE, 2019, pp. 70–77.

[3] P. J. Mucci, S. Browne, C. Deane, and G. Ho, "Papi: A portable interface to hardware performance counters," in *Proceedings of the department of defense HPCMP users group conference*, vol. 710, 1999.

[4] R. D. Falgout and U. M. Yang, "hypre: A library of high performance preconditioners," in *Computational Science — ICCS 2002*, P. M. A. Sloot, A. G. Hoekstra, C. J. K. Tan, and J. J. Dongarra, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 632–641.

[5] L. B. N. Laboratory, "Nersc's cray xc40 supercomputer," https://www.nersc.gov/users/computational-systems/cori/.

[6] J. Dongarra, M. Gates, A. Haidar, J. Kurzak, P. Luszczek, P. Wu, I. Yamazaki, A. Yarkhan, M. Abalenkovs, N. Bagherpour, S. Hammarling, J. Šístek, D. Stevens, M. Zounon, and S. D. Relton, "Plasma: Parallel linear algebra software for multicore using openmp," *ACM Trans. Math. Softw.*, vol. 45, no. 2, may 2019. [Online]. Available: https://doi.org/10.1145/3264491