Integration and Synthess for Automated Performance Tuning: the SYNAPT Project

Nicholas Chaimov, Boyana Norris, and Allen D. Malony

{nchaimov,norris,malony}@cs.uoregon.edu
Department of Computer and Information Science
University of Oregon
Eugene, OR 97403

Abstract. To address the growing needs of performance analysis and optimization of complex applications, we present our vision for a new unified architecture, SYNAPT, that can support not just data acquisition and simple analysis, but also performance knowledge preservation of models and the metadata required to make performance data analysis approaches reproducible, extensible, and composable. In addition to standard representation of performance analysis results, the SYNAPT architecture will include new functionality that enables users and tools to use and grow the comprehensive performance knowledge base at the heart of the SYNAPT performance expert system that can support both manual analysis and modeling tasks and interface with external tools such as autotuning compilers.

1 Introduction

High-performance applications and architectures are rapidly growing in scale and complexity, presenting a similarly expanding set of challenges to tools that provide performance measurement, analysis, and optimization capabilities. While many tools enable the collection of performance data at different granularities and through both non-intrusive and intrusive approaches, their analysis capabilities lag behind in functionality, scalability, usability, and interoperability. Consider, for example, performance analysis that generates some results summarizing properties of interest from the raw performance data. At present, most tools simply present the results in a graphical interface or at best store them by using a custom format, which no other tool can read. This hampers the ability to share or even just reproduce analysis results. Furthermore, other independently developed tools, such as autotuners, cannot easily use any of the derived performance metrics to help better define the potentially huge space of optimization options. To address these and other shortcomings of the existing state-of-the art performance modeling, analysis, and optimization environments, we introduce our vision for *SYNAPT* (Synthesized tools for Automated Performance Tuning), a unified architecture for *performance knowledge* preservation, integration, and evolution that enables users and tools to store and share the results of program and code analysis, architecture and systems characterization, performance experiments, modeling activities, and tuning parameters and transformations.

2 Related Work

The concept of integrated measurement and analysis can be found in several parallel performance tools projects [9, 16, 21]. However, these environments generally are not extensible enough to accept input of more information from other sources, or do not provide interfaces usable from autotuning frameworks. In contrast, there have been several projects integrating data mining and machine learning techniques with the selection of compiler optimizations based on static properties of the code, hardware counters sampled during runtime, or both, using a variety of classifiers [5, 8, 18, 19, 22].

A major project combining machine learning for selecting compiler optimizations with a centralized database is the Collective Tuning project [7]. That project uses a modified version of GCC in which functions of interest are cloned at compile time, creating an optimized and unoptimized version. When the original function is invoked at runtime, one of the clones is selected for execution according to a probability distribution, which is initially uniform. Runtime sampling determines whether the proportion of time spent in the variants differs from the probability distribution. If the proportion of time spent in a function is significantly lower than the expectation assuming equality, then it can be inferred that that variant offers better performance than its competitor. The result of the competition is then communicated to a central server which updates probability distributions: one probability distribution for the program, another which aggregates data for programs with similar reactions to optimizations, and a third which aggregates data across all programs.

Cavazos proposed an *intelligent compiler* [4] which features a centralized knowledge base. Such a compiler would use machine learning techniques to generate performance models for predicting likely-good optimizations based upon static analysis and empirical performance measurement of both the code now being compiled as well as similar programs previously compiled, and microbenchmarks of the target system.

3 Proposed Methodology: A Knowledge Database

The idea for our proposed system comes from our experience with the TAU Performance System® [21], and in particular the included performance data management system, *TAUdb* [13], which stores profiles from performance experiments along with metadata describing the execution environment and application-specific metadata. Data stored in TAUdb is accessible from *ParaProf* [3], TAU's parallel profile analyzer, and *PerfExplorer* [12, 14], TAU's performance data mining framework. PerfExplorer provides a library of internal analysis routines and a means to incorporate statistics and data mining packages, such as R and Weka. An API to the analysis library and a Python scripting engine are available so that analysis pipelines can be specified programattically. TAUdb has been previously used to store annotated performance profiles of code variants generated during autotuning runs and to provide information on past runs in order to select good starting points for hill-climbing search algorithms in autotuning [6, 26].

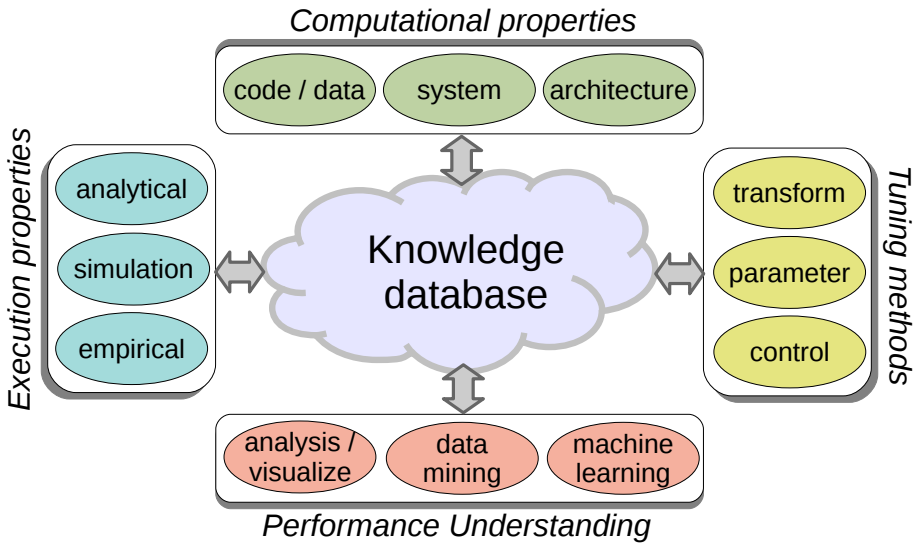

Fig. 1. The proposed system features a centralized knowledge store.

However, the existing TAUdb system is limited in that it only stores performance profiles and associated metadata. The remainder of this section describes our plans for a unified knowledge database capable of storing data from and providing data to a much broader set of tools. Figure 1 shows the central role this knowledge database plays in the SYNAPT system.

The goal of the knowledge database will be to store more comprehensive data about the **environment** in which tuning is taking place: what code is being tuned, what data are inputs to the code, what runtime systems are being used (e.g., threading libraries, network stack), and what architectures are being used (e.g., CPUs, GPUs, other accelerator cards). This is already possible to some extent with the existing TAUdb, but we envision significantly more detailed knowledge about algorithms and libraries. This will require new ways to represent this information.

3.1 Functional Descriptions

Our knowledge database should annotate code with a **description of the algorithm** the code implements, which would allow the system to identify relevant models and alternate implementations of the algorithm. While the user could manually annotate their functions, it would be better to train classifiers to identify algorithms based on features from static or dynamic analysis, as has been done recently with decision trees [24, 25]. The knowledge database would also contain data on commonly-used libraries, tagging functions in those libraries according to what algorithms they implement in the same way. The system could

then propose optimized library versions of code the user had previously developed manually. When input code already uses a library known to the system, it could provide guidance about best practices for the use of that library.

We have promising preliminary results from constructing and using such functional taxonomies for numerical libraries in the Lighthouse project [15]. For example, we have successfully integrated all of LAPACK [1] and portions of PETSc [2] and SLEPc [11] and will continue to expand with more numerical packages. The taxonomy allows users to quickly find the best method available for a given problem based on different types of search interfaces (from guided question-answer based ones for beginners, to keyword domain-specific search methods). In addition to functional descriptions, we are integrating performance models (generated through various machine learning approaches), which can enable users to discover solution methods that fit both functional and performance requirements. However, because of the lack of a common, portable, compact representation of performance analysis results, we are unable to effectively use some of the available model generation approaches (e.g., one cannot even store a DynaTree model [23] generated in R – it is only available in a single R session).

3.2 Performance Analysis Results

The knowledge database will store **performance models**, whether generated from empirical performance measurements, from simulation, or analytical models developed by users. The system could make use of these models for performance prediction in order to carry out guided search for autotuning, and could automatically generate experiments to determine whether a given model's predictions are accurate for a given code, dataset and environment.

The knowledge database will interface with **analysis tools**, such as visualization, data mining, and machine learning packages. For example, autotuning runs produce large amounts of data, so such tools are very useful in understanding the performance consequences of the various transformations which were tried. A standardized model for storing the results of such analyses in the database will be developed, so that the results can be persisted across trials and shared with other users. While the languages and approaches used for the model generation can vary, e.g., Java or Jython in PerfExplorer can be used to analyze performance data stored in TAUdb through interfaces to Weka and other data mining packages, the resulting performance metrics or models must be stored by using a common format, which does not yet exist and must be defined in order to make analyses reproducible, reusable, extensible, and composable. Such analysis results or models will be stored with provenance metadata describing the data which was analyzed and the experiments which generated the data.

Search Capabilities The knowledge database will support advanced search capabilities beyond the low-level database queries available today in TAUdb and other performance databases. Because it will also store performance models, it will be possible to use the knowledge database to search for *similar* types of

computations without knowing a priori what they are. New tools for such an expert system will rely on the database to discover potentially similar performance data *across* applications, something that is not possible to do automatically with the state-of-the-art performance tools today. Any discovered similarities can in turn be used to enrich the knowledge database with such associations, which can be used to expand the set of known computational patterns and common performance characteristics (or problems).

3.3 Integration with Autotuning Systems

Performance tuning tools today, including Orio, typically operate in a standalone fashion, i.e., given a code instance and a set of input parameters, they apply transformations and explore the space of possible transformations by using a number of optimization strategies (e.g., genetic algorithms, simulated annealing, Nelder-Mead-based optimization methods). Because purely empirical autotuning relies on compiling and executing optimized code variants in order to evaluate the objective function (e.g., time), the autotuning process can be costly in terms of time and computational resources.

The knowledge database will provide input into, and collect metadata from **autotuning systems**. It will be used to propose parameters to code transformations in source-to-source autotuning systems such as Orio [10], and will capture parameters and generated code variants as metadata alongside the original code, runtime system and architectural data described above. If the system carries out algorithm recognition as described above, it can carry out transformations which involve replacing the implementation of the algorithm, as guided by models and data from past autotuning runs. The knowledge database could also be accessed by runtime-adaptive systems, which could deposit performance data while a program is running and retrieve data used to select an algorithm or other relevant parameters.

We have previously used machine learning techniques for selecting good starting points for search in autotuning problems and for runtime selection from a library of specialized, autotuned code variants [6]. We used annotated performance data from autotuning runs across different CUDA targets to learn decision trees. Using a decision tree learned from a set of such devices to predict code transformation parameters for another device not previously tested significantly reduced the number of evaluations and overall time spent in search on that device, while still producing the same output code as a search from a default configuration. By autotuning matrix kernels across matrices of different sizes, we produced a library of size-specialized kernels. This system is shown in Figure 2. A specialized kernel could be selected by evaluating the decision tree at runtime when a wrapper kernel was invoked. We have recently added OpenCL code generation support to Orio, allowing a much wider range of possible target architectures, making the problem of selecting good initial parameters more difficult; a system combining profile data with architectural models and code characteristics would help with this.

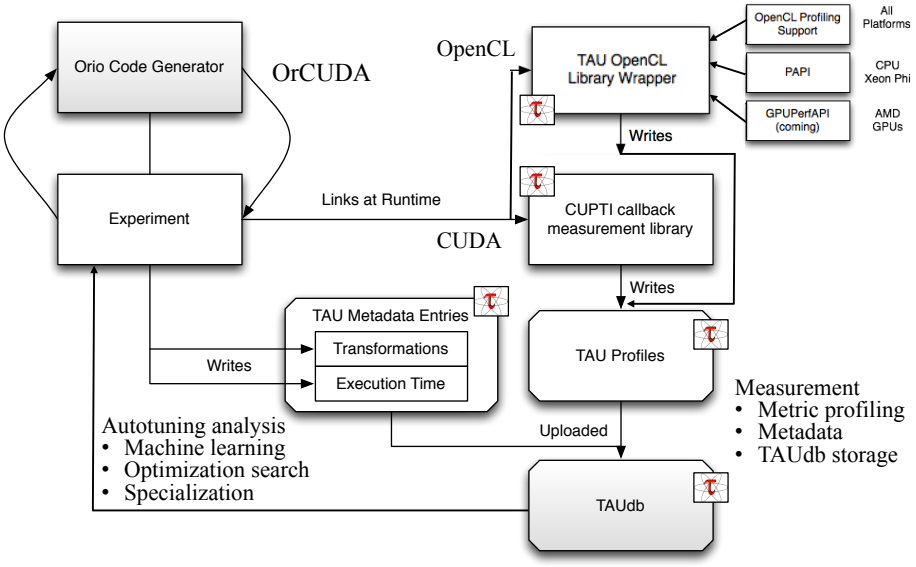

Fig. 2. Architecture of the existing system integrating TAU and Orio. The system stores annotated performance profiles in a database, which can be used to select starting points for autotuning experiments.

In addition to enabling the better selection of tuning parameters, the knowledge database and associated analysis infrastructure can support autotuning by providing models that can be used to guide the *transformations* themselves. We envision an approach that builds on the successful experience of autotuning compilers such as MilepostGCC [17, 20], which considers a set of features corresponding to program entities (e.g., functions, instructions and operands, variables, types, constants, basic blocks, and loops) and the relations among them (e.g., call graph, control-flow graph, loop hierarchy, control dependence graph, and data dependence graph). Unlike general compiler approaches, however, Orio can also capture some higher level, possibly domain-specific information (e.g., stencil-based computations, linear algebra operations), which can be used to produce more accurate models of numerical kernels or algorithms. Given such models, autotuners such as Orio would no longer require the user to explicitly specify the set of transformations to perform but would instead select them based on the models created by analyzing the results from previous autotuning experiments in the knowledge database.

4 Conclusion

High-performance parallel computing is evolving towards systems of greater complexity, increasingly challenging our abilities to create HPC software productively

and effectively. Current parallel performance tools are necessary for understanding performance characteristics and inefficiencies, but they are insufficient by themselves to address complex optimization. On the other hand, autotuning frameworks are eager to obtain richer performance information wherever available and support within the tool environments for automated experimentation, data and results management, and sophisticated analysis. Our SYNAPT concept identifies the importance of knowledge integration as the key factor for tool synthesis, both with respect to information sharing and tool interoperation. The general idea is to shift the focus of attention to what tools are producing as "knowledge" and in what forms so that other tools can use the collective knowledge base for enabling greater functionality.

We have taken initial steps towards the SYNAPT architecture with the Orio autotuning framework and TAU. Our early experiences of multi-target autotuning for accelerators support the benefits of improved automation, data mining, and association of tuning context with results, to aid in learning features and correlations. However, it is apparent from this work that the real challenge for SYNAPT research will be in formulation of techniques and methods for knowledge representation, management, and sharing.

References

1. LAPACK - Linear Algebra PACKage, 2014.
2. Balay, S., Brown, J., Buschelman, K., Gropp, W. D., Kaushik, D., Knepley, M. G., McInnes, L. C., Smith, B. F., and Zhang, H. PETSc Web page, 2013. http://www.mcs.anl.gov/petsc.
3. Bell, R., Malony, A., and Shende, S. A portable, extensible, and scalable tool for parallel performance profile analysis. In *European Conference on Parallel Processing (EuroPar 2003)* (Sept. 2003), vol. LNCS 2790, pp. 17–26.
4. Cavazos, J. Intelligent compilers. In *Cluster Computing, 2008 IEEE International Conference on* (Sept 2008), pp. 360–368.
5. Cavazos, J., Fursin, G., Agakov, F., Bonilla, E., O'Boyle, M. F. P., and Temam, O. Rapidly selecting good compiler optimizations using performance counters. In *Proceedings of the International Symposium on Code Generation and Optimization* (Washington, DC, USA, 2007), CGO '07, IEEE Computer Society, pp. 185–197.
6. Chaimov, N., Biersdorff, S., and Malony, A. D. Tools for machine-learning-based empirical autotuning and specialization. *International Journal of High Performance Computing Applications 27*, 4 (2013), 403–411.
7. Fursin, G., and Temam, O. Collective optimization: A practical collaborative approach. *ACM Trans. Archit. Code Optim. 7*, 4 (Dec. 2010), 20:1–20:29.
8. Ganapathi, A., Datta, K., Fox, A., and Patterson, D. A case for machine learning to optimize multicore performance. In *Proceedings of the First USENIX conference on Hot topics in parallelism* (Berkeley, CA, USA, 2009), HotPar'09, USENIX Association, pp. 1–1.
9. Geimer, M., Wolf, F., Wylie, B. J. N., and Mohr, B. Scalable parallel trace-based performance analysis. In *Proceedings of the 13th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface (2006)* (Bonn, Germany, 2006), pp. 303–312.

10. Hartono, A., Norris, B., and Sadayappan, P. Annotation-based empirical performance tuning using Orio. In *Proceedings of the 23rd IEEE International Parallel & Distributed Processing Symposium* (Rome, Italy, 2009).
11. Hernandez, V., Roman, J. E., and Vidal, V. SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems. *ACM Trans. Math. Software 31*, 3 (2005), 351–362.
12. Huck, K., and Malony, A. PerfExplorer: A performance data mining framework for large-scale parallel computing. In *Supercomputing Conference (SC 2005)* (Nov. 2005), ACM.
13. Huck, K., Malony, A., Bell, R., and Morris, A. Design and implementation of a parallel performance data management framework. In *International Conference on Parallel Processing (ICPP 2005)* (Aug. 2005), IEEE Computer Society.
14. Huck, K., Malony, A., Shende, S., and Morris, A. Knowledge support and automation for performance analysis with PerfExplorer 2.0. *The Journal of Scientific Programming 16*, 2-3 (2008), 123–134. (special issue on Large-Scale Programming Tools and Environments).
15. Lighthouse project. https://code.google.com/p/lighthouse-taxonomy/, 2013.
16. Mellor-Crummey, J. HPCToolkit: Multi-platform tools for profile-based performance analysis. In *5th International Workshop on Automatic Performance Analysis (APART)* (November 2003).
17. MILEPOST GCC: Collaborative development website. http://cTuning.org/milepost-gcc, 2014.
18. Monsifrot, A., Bodin, F., and Quiniou, R. A machine learning approach to automatic production of compiler heuristics. In *Proceedings of the 10th International Conference on Artificial Intelligence: Methodology, Systems, and Applications* (London, UK, UK, 2002), AIMSA '02, Springer-Verlag, pp. 41–50.
19. Murthy, G., Ravishankar, M., Baskaran, M., and Sadayappan, P. Optimal loop unrolling for GPGPU programs. In *Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on* (2010), pp. 1–11.
20. Namolaru, M., Cohen, A., Fursin, G., Zaks, A., and Freund, A. Practical aggregation of semantical program properties for machine learning based optimization. In *Proceedings of the 2010 International Conference on Compilers, Architectures and Synthesis for Embedded Systems* (New York, NY, USA, 2010), CASES '10, ACM, pp. 197–206.
21. Shende, S., and Malony, A. TAU: The TAU parallel performance system. *International Journal of High Performance Computing Applications 20*, 2 (2006), 287–311.
22. Stephenson, M., and Amarasinghe, S. Predicting unroll factors using supervised classification. In *Proceedings of the International Symposium on Code Generation and Optimization* (Washington, DC, USA, 2005), CGO '05, IEEE Computer Society, pp. 123–134.
23. Taddy, M., Gramacy, R., and Polson, N. Dynamic trees for learning and design. *Journal of the American Statistical Association 106*, 493 (2011), 109–123.
24. Taherkhani, A. Using decision tree classifiers in source code analysis to recognize algorithms: An experiment with sorting algorithms. *The Computer Journal* (2011).
25. Taherkhani, A. *Automatic Algorithm Recognition Based on Programming Schemas and Beacons: A Supervised Machine Learning Classification Approach*. PhD thesis, Aalto University, Esbo, Finland, 2013.
26. Yifan, Z. Extensions of an empirical automated tuning framework. Master's thesis, University of Maryland, 2013.