Enhancing Learning-based Autotuning with Composite and Diagnostic Feature Vectors

Saami Rahman
Texas State University
San Marcos, TX
saami.rahman@txstate.edu

Richard Hay
Texas State University
San Marcos, TX
rh1357@txstate.edu

Mario A Gutierrez
Texas State University
San Marcos, TX
mag262@txstate.edu

Apan Qasem
Texas State University
San Marcos, TX
apan@txstate.edu

1. INTRODUCTION

The use of machine learning techniques has emerged as a promising strategy for model-based tuning of HPC applications. Central to the success of such learned heuristics is the construction of a summary feature vector that accurately captures program behavior. Although the salient features of certain domain specific kernels (e.g., linear algebra) are well understood, at least in principle, automatically deriving suitable features for general numerical applications continues to be a significant challenge. Due to the size and complexity of programs, theoretically there are an infinite number of potential features to choose from. Furthermore, since performance influencing parameters for an application can vary across architectures, different objectives, and even separate runs with different input sets, it becomes critically important to include in the feature vector, dynamic attributes extracted during program execution.

A survey of major efforts in ML-based autotuning reveals the use of 200 distinct program features [1, 2, 3, 4, 5, 6, 7, 10, 11]. Although these features capture many of the performance characteristics of a program, they tend to have three limitations. The features are

- typically *hand picked* by experts. Although effective in some situations, this ad-hoc approach can be dangerous because not all attributes that influence the outcome vector may be known and irrelevant features may be introduced that can weaken the predictive capabilities of the learned heuristic
- mostly *rudimentary* and therefore do not always correlate to broader performance issues that can be directly addressed by high-level code transformations
- *static* and hence do not capture the dynamic behavior of the program in the target environment

This paper addresses each of these issues and introduces two new classes of enhanced features that accurately characterize program behavior across architectures and increase the efficacy of supervised learning algorithms when applied within an autotuning system.

SC'13 Nov 17-22, Denver, Colorado, USA

2. ENHANCED FEATURE VECTORS

We briefly discuss the two new classes of features next.

2.1 Composite Features

We propose the inclusion of a new class of composite features that can be extracted using a combination of source code analysis and execution profiling. The key idea is to combine several elementary program attributes to produce a quantifiable feature value that is more directly linked to performance. The composite or *macro* features we construct are typically utilized by sophisticated compiler heuristics in making decisions about *how*, *where* and *if* an optimization should be applied. For instance, the key attribute for tiling a loop nest is the size of the working set. Although a property such as the working set is essential for characterizing performance on current architectures they are not captured by elementary program features because their derivations require explicit analyses. In our system, we build on a series of compiler models to derive a collection of these composite features. Specifically, the features address the three core performance imperatives for current multicore architectures: (i) exploit parallelism at different levels (e.g., reduce communication cost, maintain sufficient work per thread); (ii) use and share memory resources intelligently (e.g., exploit data reuse, eliminate cache contention); and (iii) utilize specialized hardware constructs (e.g., use vector unit, use hardware prefetcher). A partial list of the composite features currently supported by our system is given below.

- reuse distance profiles (expressed as a histogram)
- working set size
- thread granularity
- amount of data reuse at loop level
- register pressure of inner loops
- stride access information
- prefetch candidacy

2.2 Diagnostic Features

For the second class of program features, we shift our attention from program characteristics to performance characteristics. Program features, be it static or dynamic, are typically architecture-independent and hence do not provide

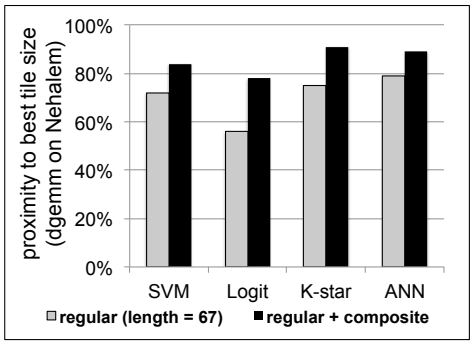

Figure 1: Effect of composite features on optimal tile size selection

insight about performance portability of a given program. We incorporate in the feature vector a series of diagnostic dynamic features derived from fine-grain HW performance counter readings collected during program execution. A diagnostic feature is one that identifies and characterizes the *causes* rather than the symptoms of performance loss. For instance, a high L1 cache miss rate is a symptom of performance while *a conflict between two cached data structures* might be the cause of the problem. When used as a feature, a *cause* delivers significantly more information to the learning algorithm than a symptom.

To obtain these diagnostic features, we take advantage of the precise event based sampling (PEBS) and instruction based sampling (IBS) on the latest generation of Intel and AMD processors, where the *effective data address*, along with other information, is made available for each sampled event. The availability of the data address is particularly significant because these addresses can be collected to construct a range of diagnostics, including the following

- cache classification misses : compulsory, capacity and conflict
- cache coherence and sharing : true and false sharing, replication, pollution
- cost of inter-thread synchronization
- hardware prefetch activity

3. STATUS

Currently we have several composite and diagnostic features implemented in our ML-driven autotuning system [9]. Results from our preliminary evaluation are very encouraging. Selected results are presented in the accompanying poster and further details can be obtained in a Technical Report [8].

In the interest of space, in this abstract, we highlight results from a single study. In this study, four supervised learning algorithms - Support Vector Machines (SVM), Logistic Regression, K-star and Artificial Neural Networks - were trained with two sets of feature vectors: one set included 67 static features collected from source code while the other included a composite feature (working set) in addition to the static features. The algorithms were then used to predict the optimal tile size on several platforms with different cache configurations. As seen in Fig. 1, the results demonstrate a clear advantage for the *better-informed* models with as much as a 23% increase in prediction accuracy.

Acknowledgement

This work was funded by the National Science Foundation under award CNS-1253292.

References

[1] F. Agakov, E. Bonilla, J. Cavazos, B. Franke, G. Fursin, M. O'Boyle, J. Thomson, M. Toussaint, and C. Williams. Using machine learning to focus iterative optimization. In *International Symposium on Code Generation and Optimization, 2006. (CGO 2006).*, New York, NY, 2006.

[2] R. Cammarota, A. Kejariwal, D. Donato, A. Nicolau, and A. Veidenbaum. Selective search of inlining vectors for program optimization. In *Proceedings of the 9th conference on Computing Frontiers*, CF '12, pages 257–260, 2012.

[3] J. Cavazos, G. Fursin, F. Agakov, E. Bonilla, M. F. P. O'Boyle, and O. Temam. Rapidly selecting good compiler optimizations using performance counters. In *CGO '07: Proceedings of the International Symposium on Code Generation and Optimization*, pages 185–197, Washington, DC, USA, 2007. IEEE Computer Society.

[4] K. Datta, M. Murphy, V. Volkov, S. Williams, J. Carter, L. Oliker, D. Patterson, J. Shalf, and K. Yelick. Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures. In *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 1–12, Piscataway, NJ, USA, 2008. IEEE Press.

[5] G. Fursin, Y. Kashnikov, A. W. Memon, Z. Chamski, O. Temam, M. Namolaru, E. Yom-Tov, B. Mendelson, A. Zaks, E. Courtois, F. Bodin, P. Barnard, E. Ashton, E. Bonilla, J. Thomson, C. Williams, and M. O'Boyle. Milepost gcc: machine learning enabled self-tuning compiler. *International Journal of Parallel Programming*, 39, 2011.

[6] X. Li and M. J. Garzaran. Optimizing matrix multiplication with a classifier learning system. In *Languages and Compilers for Parallel Computers (LCPC05)*, 2005.

[7] S.-w. Liao, T.-H. Hung, D. Nguyen, C. Chou, C. Tu, and H. Zhou. Machine learning-based prefetch optimization for data center applications. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pages 56:1–56:10, 2009.

[8] A. Qasem and S. Chen. Using macro features in learning algorithms for optimizing dense-matrix computations. Technical Report CSTR12-17, Dept. of Computer Science, Texas State University, Jan. 2012.

[9] S. Sarangkar and A. Qasem. Mats: A model-driven adaptive tuning system for parallel workloads. *Journal of Parallel and Cloud Computing (JPCC)*, 1(2), 2012.

[10] M. Stephenson and S. Amarasinghe. Predicting unroll factors using supervised classification. In *CGO*, San Jose, CA, USA, March 2005.

[11] K. Stock, L.-N. Pouchet, and P. Sadayappan. Using machine learning to improve automatic vectorization. *ACM Trans. Archit. Code Optim.*, 8(4):50:1–50:23, Jan. 2012.