Improving Autotuning Efficiency and Portability Through Feedback Diagnostics*

Qing Yi[1] Santosh Sarangkar[2] Apan Qasem[2]

[1] University of Texas at San Antonio
[2] Texas State University

1 The Role of Feedback

Enhancing both the efficiency and portability of empirical tuning is critically important for existing autotuning systems to successfully explore an exponentially growing search space of code optimizations. The quality of feedback plays a key role in this success. The right choice and use of feedback metrics can help diagnose performance problems quickly and lead to better solutions. Exploiting feedback for faster tuning involves answering several challenging questions:

- How to collect sufficient information so that performance bottlenecks can be easily identified? How to build performance models that effectively correlate different interacting factors of architectural components as well as code optimizations to identify *causes* of performance problems?
- How to use enhanced search efficiency to more effectively solve NP-complete problems faced by traditional compilers? How to enable portability of tuning so that results of tuning similar applications or architectures in the past are used to drive more efficient tuning of new applications/architectures?

2 Diagnosing Performance Bottlenecks

Feedback diagnostics in autotuning systems can be enhanced in several ways

1. Improve granularity by collecting measurements at loop-level [1] and increase volume by utilizing available hardware performance counters [2].
2. Synthesize multiple performance metrics to create feedback that can diagnose causes of performance bottlenecks.

The use of synthesized or derived metrics can provide key insight into application performance characteristics specific to a target architecture. When utilized by a search engine, this information can significantly enhance search efficiency. As an example, consider the search space of a 7-point stencil kernel on a quad-core platform. Fig. 1(a) shows the execution time of different thread schedules and Fig. 1(b) shows the *same* search space sorted by hardware prefetch activity

* This work is funded by the National Science Foundation under Grant No. 0833203 and No. 0747357 and by the Department of Energy under Grant No. DE-SC001770.

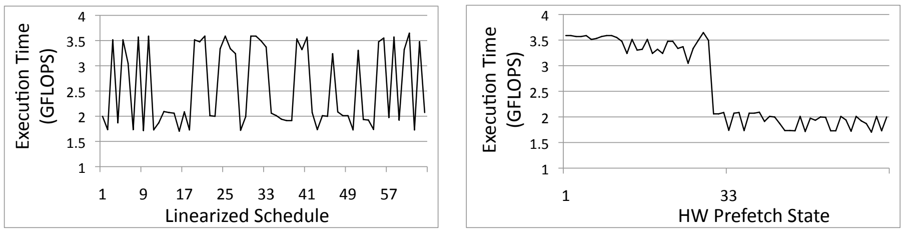

Fig. 1. `advect3d` search space

(a synthesized metric). Although the search space of different thread schedules appears to be quite random, a clear pattern emerges when feedback is augmented with HW prefetch information. When exploited by the search algorithm, the improved diagnostic leads to an overall 70% reduction in tuning time.

3 Improving Tuning Efficiency And Portability

The tuning space grows exponentially as each additional code optimization is parameterized. Few existing research have investigated how to handle a potential search space explosion when architectural optimizations at different levels, e.g., blocking, unroll-and-jam, register allocation, instruction scheduling, are extensively parameterized. Compilers employ heuristics to make optimization decisions precisely because trying out all the configurations is NP-complete. However, can auto-tuning be used to help solve NP-complete problems?

To answer these challenges, the efficiency and effectiveness of auto-tuning must be immensely improved. In particular, machine learning techniques and new performance models need to be developed so that tuning data collected in the past can be used to guide better decisions in the future [3], The search process needs to be parallelized so that different optimization configurations can be explored simultaneously. The advent of the multi-core era offers a great opportunity for more effectively solving NP-complete problems in compilers.

References

1. D. H. Bailey, J. Chame, C. Chen, J. Dongarra, M. Hall, J. K. Hollingsworth, P. Hovland, S. Moore, K. Seymour, J. Shin, A. Tiwari, S. Williams, and H. You. Harnessing the power of emerging petascale platforms. *Journal of Physics: Conference Series*, 2007.
2. 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.
3. C. Dubach, T. M. Jones, E. V. Bonilla, G. Fursin, and M. F. P. O'Boyle. Portable compiler optimisation across embedded programs and microarchitectures using machine learning. In *MICRO 42: Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 78–88, New York, NY, USA, 2009. ACM.