Evaluating Autotuning Heuristics for Loop Tiling

Extended Abstract

Tomoya Yuki
Tokyo Institute of Technology
Tokyo, Japan
yuki.t.ab@m.titech.ac.jp

Yukinori Sato
Tokyo Institute of Technology
Tokyo, Japan
yukinori@el.gsic.titech.ac.jp

Toshio Endo
Tokyo Institute of Technology
Tokyo, Japan
endo@is.titech.ac.jp

ABSTRACT

Loop tiling is known as a technique that uses cache hierarchy efficiently and improves performance since long ago. As many-core processors become popular more and more for parallel computing, tile sizes need to adjust more technically. To tackle with this situation, we develop an auto-tuning tool for tile size optimization based on polyhedral compilation, and implement our original heuristic method. It iterates measurement of binary built with each tile size under specific scheduling. We evaluate our method and two existing heuristics known as general purpose methods. From the result, we demonstrate that our method obtains good tile sizes with shorter search steps and higher accuracy than the other methods.

KEYWORDS

loop tiling, auto-tuning

1 INTRODUCTION

Loop tiling is a well-known technique to use cache hierarchy efficiently. It divides loop iterations into smaller blocks and adjusts their schedule. The number of the blocked loop iterations is called tile size, which is usually small. These tile sizes need to be adjusted specifically for each system. If they are proper sizes, loop tiling enhances data reuse and reduces cache hit misses. As a result, performance becomes better than the one without tiling. This loop tiling takes much time and laborious effort if it is performed manually. To solve this issue, an approach that automates it using polyhedral compilers is becoming feasible. Inside a polyhedral compiler, polyhedral model is used, which mathematically represents loop conditions and shapes. We use Polly[1] as a polyhedral compiler in this paper.

Recently, many-core processors are commonly used as an attractive architecture in the HPC field. This time we adopt Xeon Phi as a representative many-core processor. In that case, tile size adjustment is not simple and easy. For example, TurboTiling[3] which is a static calculation approach does not consider about too many core processors, so that we cannot apply simply the algorithm to such systems.

2 OUR AUTO-TUNING TOOL AND SEARCH METHOD

We focus on the problem between adjusting tile sizes and using many-core processors. A dynamic approach takes more time than a static one, however higher accuracy. We develop an auto-tuting tool for loop tiling named PATT (Polyhedral compilation based AuTo Tile size optimizer). It works like a compiler driver, though it needs some user setting values for tile size selection. This tool includes "I-PATT" method which consists of our smart scheduling algorithm specific for tile size adjustment.

I-PATT separates handling triply nested loop and doubly nested one. The reason is that we have already known the outer-most tile size is the most important for triply nested loop because load balance is the dominant factor for performance. I-PATT considers this at first, so fixes inner tile sizes to 32 and iterates compilation and measurement some times changing the outer-most tile size. When it finishes, I-PATT starts to search the good sizes of inner tile on fixed the outer-most tile size. At first the search space is wide and coarse, and it gradually becomes short and fine. The search for a space is based on hill climbing algorithm. There are two reasons that we adopt this customized search: to reduce the search time and to move over a space of similar performance. From some preliminary experiments, this customized search algorithm works well particularly for tile size adjustment.

3 THE EXISTING SEARCH METHODS

We prepare two other methods for comparison to our method, which are a Simulated Annealing (SA) method and a Nelder-Mead Simplex (NM) method. They are ported and implemented from Orio[2] that is another auto-tuning tool for general purpose into our tool.

SA is a method based on local search and takes a probabilistically technique using a "Temperature" idea. It iterates testing and moving states reducing temperature gradually. Even if the performance of the next state is worse than the current one, SA moves the state probabilistically using temperature value in order to get out from a local best. By the way, in the case of tile size adjustment, search methods must consider integer problem and multiple dimension. SA defines "neighbor" as all indexes in the range of "neighbor distance", which is fixed 1. When SA moves a state, it picks up randomly one of points in neighbor. Finally, when temperature reduces enough, SA ends.

NM is a method based on a concept of moving a "Simplex", which consists of N coordinates. N is the number of dimension +1. For example, a two-dimensional problem needs three coordinates as a simplex. First, NM decides an initial simplex and tests all coordinates in the one. Then, it changes the simplex shape step by step using the following functions:

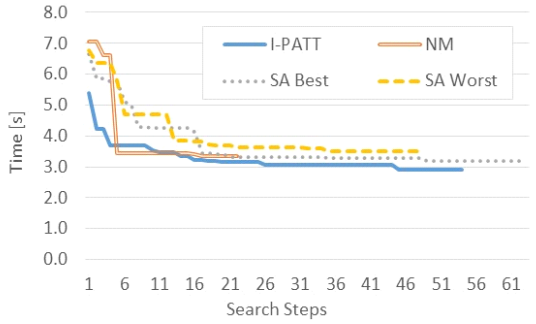

Figure 1: Convergence speeed of *gemm* kernel

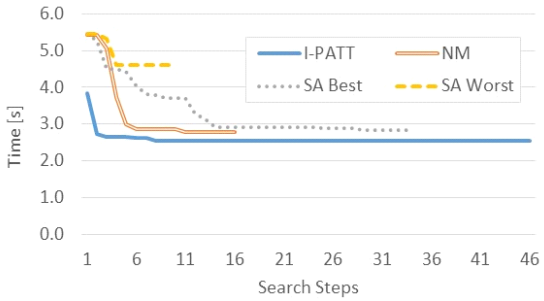

Figure 2: Convergence speed of *atax* kernel

Table 1: Final results of each method

	Search Method	Tile Sizes	Time	%
gemm	Brute-Force	16, 1124, 12	2.89	100.0
	I-PATT	16, 1192, 12	2.91	99.3
	NM	16, 36, 48	3.36	86.0
	SA (worst)	16, 28, 44	3.50	82.6
	SA (best)	16, 24, 80	3.19	90.6
atax	Brute-Force	8, 1940	2.50	100.0
	I-PATT	8, 2096	2.53	98.8
	NM	8, 80	2.78	89.9
	SA (worst)	36, 36	4.60	54.3
	SA (best)	8, 64	2.84	88.0

Reflection, Expansion, Contraction, Shrink. As a result, the simplex comes near to the best coordinates quickly in theory.

4 EVALUATION

We adopt Polybench[4] as a benchmark. Evaluation of kernels in this benchmark has an affinity with Polly used in PATT. We pick up *gemm* and *atax* kernels as representatives using triply nested loop and doubly nested one, respectively.

Our environment for evaluation is the following. We use Intel Xeon Phi Processor 7210 with 64 threads. Data size is "LARGE" on Polybench. The initial coordinates of SA and NM are "all32". This means all tile sizes is 32, which is empirically known as a good parameter. We note that a few user setting values of each method are optimized based on the results of our preliminary experiments.

Figure 1 shows the comparison of each method on *gemm* kernel. "Search Steps" are the actual numbers of building and measuring binaries. This figure indicates that NM is reaches a fast state quickly, however stops early so it cannot explore further into better state. SA's convergence speed is slower than other methods. I-PATT is usually the best speed and accuracy. Figure 2 shows the same comparison on *atax*. SA (worst) remains a bad state in the early step because SA fits into a local optimum when the generated random number sequence badly affects performance. NM is better than SA, and I-PATT is the best. Table 1 shows final results of each method. "Brute-force" is an exhaustive experiment. This table indicates that I-PATT reaches almost 99% of the fastest performance while SA and NM remain at most 90%.

We consider there two reasons for the low accuracy of SA and NM. First, the outer-most tile size is very sensitive for performance than the other dimension tile sizes. A small data size and a bunch of threads cause load imbalance, which is the dominant factor for performance. Search method should separate handling the outer-most loop tile size adjustment and inner ones like I-PATT. Second, inner loop tile sizes are usually insensitive for performance in wide neighbor ranges. Thus, it is needed to perform a wide and coarse search at first.

5 CONCLUSION

In this paper, we have focused on importance of loop tiling optimization on many core processors. We have developed an auto-tuning tool and implemented our original search algorithm, I-PATT, which is specific for tile size adjustment. We have evaluated I-PATT, SA and NM methods. From the results, we have confirmed that I-PATT obtained the almost best tile sizes successfully, and SA and NM also worked well up to a certain performance.

We have found from this research that general purpose auto-tuning methods like SA or NM have limits of convergence speed and accuracy. It needs some specific techniques for the problem of tile size adjustment on many core processors. Considering load balance and wide-and-coarse search are important even if we use general purpose methods. As a future work, we plan to improve I-PATT method to work more generically for many other kernels.

REFERENCES

[1] Tobias Grosser, Armin Groesslinger, and Christian Lengauer. 2012. Polly - Performing polyhedral optimizations on a low-level intermediate representation. *Parallel Processing Letters* 22, 04 (2012), 1–28.

[2] A. Hartono, B. Norris, and P. Sadayappan. 2009. Annotation-based empirical performance tuning using Orio. In *2009 IEEE International Symposium on Parallel Distributed Processing*. 1–11.

[3] Sanyam Mehta, Rajat Garg, Nishad Trivedi, and Pen-Chung Yew. 2016. TurboTiling: Leveraging Prefetching to Boost Performance of Tiled Codes. In *Proceedings of the 2016 International Conference on Supercomputing (ICS '16)*. 38:1–38:12.

[4] Tomofumi Yuki and Louis-Noël Pouchet. 2016. PolyBench. https://sourceforge.net/projects/polybench. (2016).