An Efficient Collaborative Filtering Approach Using Smoothing and Fusing

Xin Liu, Daqiang Zhang, Jingyu Zhou, Minyi Guo, Yao Shen
Department of Computer Science
Shanghai Jiao Tong University
No. 800, Dongchuan Road, Shanghai, P.R.China
{navyliu, zhangdq}@sjtu.edu.cn, {guo-my, zhou-jy, shen_yao}@cs.sjtu.edu.cn

***Abstract*—In advent of multicore era, plain C/C++ programming language can not fully reflect computer architectures. Source-to-source transformation helps tailor programs close to contemporary hardwares. In this paper, we propose a template-based approach to perform transformation for programs with rich static information. The template metaprogramming techniques we present can conduct parallelization and memory hierarchical optimization for specific multicores. They enable programmers to utilize new architectural features and parallel patterns by extending template library. We implement a prototype template library – libvina to demonstrate the idea. Finally, We evaluate our template library on commodity x86 and GPU platforms by a variety of typical applications in multimedia and scientific fields. In experiments, we show that our approach is flexible to support multiple parallel models and capable of transforming sequential code to parallel equivalence according to specific multicore architectures. Moreover, the cost of programmability using our approach to adapt more than one multicore platform is manageable.**

I. Introduction

Multicores rely on parallelism and memory hierarchy to improve performance. Both duplicated processors and elaborated storage-on-chip require programmers to restructure their source code and keep tuning binaries for a specific target. Therefore, non-trivial knowledge of underlying machine's architectures is necessary to write high-performance applications. More worse, various implementations of multicores bring many architectural features to programmers, which in fact further enlarge the gap between software programmers and hardware vendors.

Traditionally, algorithm experts usually focus on their specific domains and have limited insights on diverging computer systems. Writing algorithms in sequence, they expect hardwares and compilers to guarantee decent performance for their programs. The expectation was roughly held until explicit parallel system was introduced to computer community. Since the frequency of microprocessor increases slowly, it has been difficult to obtain free performance improvement from hardware's refinement. Workloads of application developers surge for parallel computer systems. In essence, it is because plain C/C++ can not fully reflect contemporary parallel architectures. It is desirable to develop methods to adapt to diverging multicore architectures.

Although extensive researches on non-traditional programming models obtain fruitful achievements, mainstream software development still stay at imperative programming languages and multi-threading. We think the primary reason is software cost. Considering the time span which a large computer system serves, hardwares are cheap and become cheaper with time elapsing, while software and well-trained personnel are expensive. Because numerous legacy software systems were designed and developed in conventional programming model, vendors usually prefer to maintain and update them rather than rebuilding from scratch. Nevertheless, exploiting horsepower of multicore processors for legacy and new systems is a moderate issue.

Source-to-source transformation can help tailor specific architectures. In particular, some transformations can extend traditional programming language to support multicore architectures. OpenMP [**?**] and Sequoia [**?**], [**?**] are typical examples. One distinct advantage comparing with other fancy languages is that they support progressive parallelization from original source code, which can guarantee to keep software investment. OpenMP transform program regions into fork-join threads based on pragma directives. Sequoia attempts to map computation-intensive functions on a machine tree describing in configuration file. In those extended languages, a set of language constructs are provided to support specific parallel patterns, so they need dedicated compilers. However, the ad-hoc approaches of source-to-souce transformation can not support general parallel patterns. It is not a trivial task to determine how many language constructs should be provided by compilers to well support the full spectrum of multicores.

We present an approach to perform source-to-souce transformation to support multicore using C++ template mechanism. We uses *tasks* to abstract computation-intensive and side-effect free function. Our primary idea is to extend C++ template specialization to task for different multicores. Template classes can transform a task into other forms according the parallel patterns and then map to threads to execute in parallel.

The primary limitation is that only static information are available at compile time. Therefore, it only works for programs which own rich static information. Fortunately, applications with this characteristic are pervasive in multimedia, digital processing, and scientific computation. Because template takes effect at compile time, it is possible to avoid deploying run-time for transformation, which means that it can incur minimal runtime cost. Besides, our template-based

approach imposes fewer restricts comparing with other static approaches:

- Flexibility: We proposed a way to perform source-to-source transformation by metaprogramming. Because it can manipulate source code in metaprograms, our approach does not bind any parallel models. It is easy to change transform to fork-join, or perform computation as pipeline. In addition, our approach can deploy any thread implementations to support parallelism and concurrency. We experiment pthread, low-level threads provided by OS and device driver. As far as we know, no parallel programming language declares such flexibility. Theoretically, metaprogramming is as expressive as any general-purposed programming languages, so we think it is a promising approach to explore more parallel patterns beyond this paper.
- Extendability: It is extensible to develop new template class to utilize new architectural features and parallel patterns. Template metaprogramming is intimate for C++ programmers. It is easy to extend new execution models and parallel patterns. Other approaches have to ratify languages and then modify compiler to complete features. The progress is usually a year-old compaign and can not determined by software developers alone.
- Portability: Template is part of ISO C++ [**?**], [**?**]. Template-based approach is applicable for every platforms with standard C++ compiler. Template metaprogramming is widely used in other applications in C++ community and full-blown metaprogramming libraries like MPL [**?**] is portable. Through good encapsulation of platform details, most of code in our template approach can be reused.

The remaining parts of this paper are structured as follows. Section 2 shows techniques to perform transforms by template metaprogramming. Audiences with C++ template programming experiences or functional programming language concepts are helpful but not prerequisites. Then Section 3 presents some typical transformations by our template library. Experiments are in Section 4 to evaluate performance on both CPU and GPU. Section 5 summarizes some related works on library-based approach and language extension to support multicore architectures. The last section is conclusion and future work.

II. Template-based Programming model

Libvina framework utilizes C++ template mechanism to perform source-to-source transformation for multicores. We uses *tasks* to abstract side-effect free functions. A task is wrapped in the form of template class. *TF classes* are able to manipulate tasks, which take responsiblilty for transforming a tasks into a group of subtasks. Finally, we map subtasks into threads for specific architectures. Fig. 1 depicts the diagram of template-based programming model.

Applications using our template-based programming model are free to choose parallel pattern. An example using map/reduce model is shown in Fig **??**. A matrix-multiplication

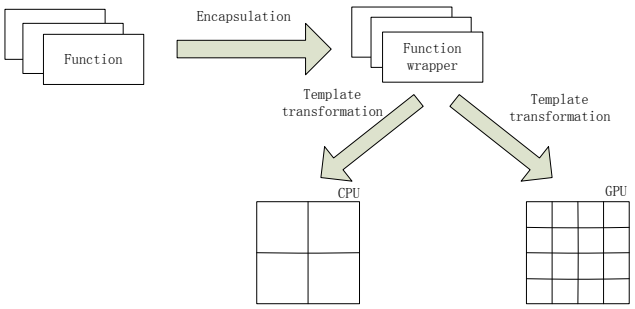

Fig. 1. template transformation

application can be divided into many submatrices multiplication and reduced them into the result. We can apply a TF class dedicated to this parallel pattern and it hierarchically generates a subtasks and reductions. The decomposition rule and termination of recursion is programmed in template metaprogramming.

Our programming model facilitates to separate two roles in software development. Algorithm-centric programmers only concern of algorithm in convential C/C++ form. They provides computation-intensive and side-effect free functions in the form of task. Otherwise, system programmers known underlying architecture are charge in developing and applying template class to specialize tasks for the concrete target. This separation not only solve the difficulties of writing and tuning multicore, but also providing a uniform programming model to develop effecient and portable parallel programs.

The primary limitation of libvina framework is that we perform transformation using template metaprorgramming. Only static information, *i.e.* static constant value and types in C++, is available at compile time. In libvina framework, we utilize array dimensions to estimate problem size and divide a task into subtasks. Thus, our programming model orients to programs which contains rich static information. Fortunately, it is not uncommon that array dimensions are determinable at compile time for embeded and scientific programs.

References

[1] Y. Koren, "Factorization meets the neighborhood: a multifaceted collaborative filtering model," in *Proc. of the 14th ACM SIGKDD*, 2008, pp. 426–434.

[2] G. Linden, B. Smith, and J. York, "Amazon.com recommendations: item-to-item collaborative filtering," *Internet Computing, IEEE*, vol. 7, no. 1, pp. 76–80, 2003.

[3] A. S. Das, M. Datar, A. Garg, and S. Rajaram, "Google news personalization: scalable online collaborative filtering," in *Proc. of the 16th Int. Conf. on World Wide Web*, 2007, pp. 271–280.

[4] B. Marlin, R. S. Zemel, S. Roweis, and M. Slaney, "Collaborative filtering and the missing at random assumption," in *Proc. of the 23rd Conf. on Uncertainty in Artificial Intelligence*, 2007.

[5] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions," *IEEE Transactions on Knowl. and Data Engineering*, vol. 17, no. 6, pp. 734–749, 2005.

[6] H. Ma, I. King, and M. R. Lyu, "Effective missing data prediction for collaborative filtering," in *Proc. of the 30th ACM SIGIR*, 2007, pp. 39–46.

[7] G.-R. Xue, C. Lin, Q. Yang, W. Xi, H.-J. Zeng, Y. Yu, and Z. Chen, "Scalable collaborative filtering using cluster-based smoothing," in *Proc. of the 28th ACM SIGIR*, 2005, pp. 114–121.

[8] J. Wang, A. P. de Vries, and M. J. T. Reinders, "Unifying user-based and item-based collaborative filtering approaches by similarity fusion," in *Proc. of the 29th ACM SIGIR*, 2006, pp. 501–508.
[9] T. Zhang and V. S. Iyengar, "Recommender systems using linear classifiers," *J. Machine Learning Research*, vol. 2, pp. 313–334, 2002.
[10] Y. Zhang and J. Koren, "Efficient bayesian hierarchical user modeling for recommendation system," in *Proc. of the 30th ACM SIGIR*, 2007, pp. 47–54.
[11] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proc. of the 10th Int. Conf. on World Wide Web*, 2001, pp. 285–295.
[12] R. Bell, Y. Koren, and C. Volinsky, "Modeling relationships at multiple scales to improve accuracy of large recommender systems," in *Proc. of the 13th ACM SIGKDD*, 2007, pp. 95–104.
[13] J. Kleinberg and M. Sandler, "Using mixture models for collaborative filtering," *J. of Comp. and Sys. Sci*, vol. 74, no. 1, pp. 49–69, 2008.
[14] M. Connor and J. Herlocker, "Clustering items for collaborative filtering," in *Proc. of SIGIR'01 Workshop on Recommender Systems*, 2001.
[15] K. Cheung, K. Tsui, and J. Liu, "Extended latent class models for collaborative recommendation," *IEEE Transactions on Systems, Man, and Cybernatics. Part A: Systems and Humans*, vol. 34, pp. 143–148, 2004.
[16] T. Hofmann, "Latent semantic models for collaborative filtering," *ACM Transactions on Information Systems*, vol. 22, no. 1, pp. 89–115, 2004.
[17] R. Jin, L. Si, and C. Zhai, "A study of mixture models for collaborative filtering," *Information Retrieval*, vol. 9, no. 3, pp. 357–382, 2006.
[18] R. M. Bell and Y. Koren, "Scalable collaborative filtering with jointly derived neighborhood interpolation weights," *IEEE Int. Conf. on Data Mining*, pp. 43–52, 2007.
[19] D. M. Pennock, E. Horvitz, S. Lawrence, and C. L. Giles, "Collaborative filtering by personality diagnosis: a hybrid memory and model-based approach," in *Proc. of 16th Conf. on Uncertainty in Artificial Intelligence*, 2000, pp. 473–480.
[20] J. D. M. Rennie and N. Srebro, "Fast maximum margin matrix factorization for collaborative prediction," in *Proc. of the 22nd Int. Conf. on Machine Learning*, 2005, pp. 713–719.
[21] R. J. Mooney and L. Roy, "Content-based book recommending using learning for text categorization," in *Proc. of the 5th ACM Conf. on Digital Libraries*, 2000, pp. 195–204.
[22] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl, "Evaluating collaborative filtering recommender systems," *ACM Transactions Inf. Syst.*, vol. 22, no. 1, pp. 5–53, 2004.
[23] R. Jin, J. Y. Chai, and L. Si, "An automatic weighting scheme for collaborative filtering," in *Proc. of the 27th ACM SIGIR*, 2004, pp. 337–344.