2018 5th International Conference on Information Science and Control Engineering

Thread-Level Speculation:Review and Perspectives

Yuxiang Li
School of Information Engineering
Henan University of Science and Technology
Luoyang, China
liyuxiang@haust.edu.cn

Zhiyong Zhang
School of Information Engineering
Henan University of Science and Technology
Luoyang, China
xidianzzy@126.com

Lili Zhang
School of Information Engineering
Henan University of Science and Technology
Luoyang, China
lillyzh@126.com

Danmei Niu
School of Information Engineering
Henan University of Science and Technology
Luoyang, China
niudanmei@163.com

***Abstract*—Thread-Level Speculation (TLS) is an automatic parallelization technique for serial programs on multi-core platforms, and permits the generation of multiple threads during compilations well as in run-time. The most obvious difference between TLS and the conventional parallelization model is that TLS can partition programs into multiple threads to be speculatively executed in the presence of ambiguous data and control dependences, while the correctness of the programs is guaranteed by run-time system. As TLS (in different ways) has become ubiquitous in today′s parallel computing, it seems worthwhile to provide a review of TLS that has evolved over the last few decades. We concentrate on a comprehensive review of models with some practical relevance together with a perspective on models with potential future relevance. This review attempts to collect, organize, and summarize the most representative publications in thread-level speculation. In chronological order, TLS evolved from Hardware Thread-Level Speculation (HTLS) to Software Thread-Level Speculation (STLS), and to Algorithm Thread-Level Speculation (ATLS). Moreover, a perspective is given to clarify the future development of TLS. Aside from presenting the models, we also refer to features, implementations, and tools.**

***Keywords*-thread-level speculation; hardware thread-level speculation; irregular programs; software thread-level speculation; algorithm thread-level speculation**

I. Introduction

Irregular programs use dynamic structures such as trees, lists and Directed Acyclic Graphs (DAGs) to solve science problems and play an important role in complex network analysis, machine learning, image processing, bioinformatics analysis, climate simulation models, and so on. These programs usually have complex control flow, irregular data access pattern and own more potential parallelism. Because the irregular programs have ambiguous control and data dependences, it is difficult to find the common parallel patterns, behaviors and semantics. As a result, they are hard to be paralleled by the conventional approaches, such as OpenMP [1], TBB [2], MPI [3] and CUDA [4]. At the same time, as the rapid development of the semiconductor technology in accordance with Moore′s Law, microprocessor architecture has entered the multi-core era and multi-core processors are now the mainstream processor architectures, offering more computing and storage resources, increasing communication bandwidth and reducing communication delay. Dedicated server (e.g., the SGI Origin [5]) that can simultaneously execute multiple parallel threads are becoming increasingly commonplace on a wide variety of scale, and even personal computers are often sold in two or four processor configurations. Although hardware techniques such as simultaneous multithreading [6](e.g., the Alpha) and single-chip multiprocessing [7](e.g., the Intel Xeon, Core, and AMD Athlon) have long been exploited to provide more computing resources for parallel processing, the greatest stumbling block in the exploitation of potential performance of irregular programs is to use software techniques to automatically divide the sequential programs into multithreads which are executed in parallel and to eliminate control and data dependences among the parallel threads.

TLS allows the compiler to automatically parallelize portions of code in the presence of statically ambiguous data dependences, thus extracting parallelism between whatever dynamic dependences actually exist at run-time. To illustrate how TLS operates, Figure 1 includes four cases. Figure 1(a) shows the sequential execution, while Figure 1(b) shows the speculative execution. Once the speculative execution fails, the speculative thread will be re-executed after the master thread, shown in Figure 1(c). When a Read-After-Write (RAW) conflict occurs, the speculative thread will rollback and again execute, as is exhibited in Figure 1(d).

II. General Introduction

A. General Introduction of Thread-Level Speculation

This section provides basic background materials on TLS and defines some terms and describes how a TLS works,

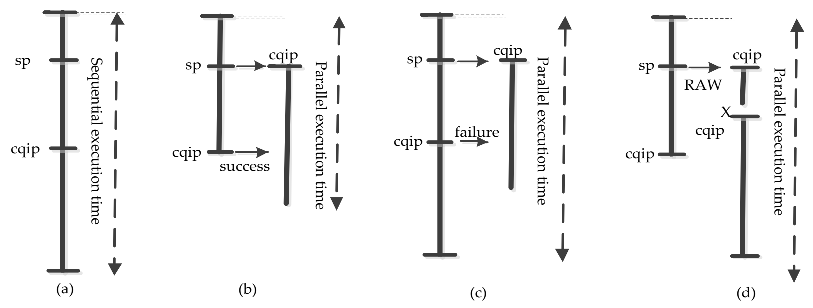

Figure 1: Thread-level speculative model

but is not a complete tutorial. Interested readers can consult References [8], [9] for more detailed background information on TLS.

TLS allows the compiler to automatically parallelize general-purpose programs by supporting the parallel execution of threads that might not actually be independent [10]. The considered issues in TLS include a thread partition scheme as well as control and data dependence elimination. Regarding the partition scheme, TLS partitions serial applications into multi-threads and speculatively executes them on the multi-cores. Of all the threads, only one is the primary thread, which has the ability to submit data, while the others are speculative threads which can commit data once being converted to be a primary thread. For thread partition, the most widely used method is to insert Spawning Point (SP) and Control Quasi Independent Point (CQIP) [11] into serial programs to form various threads so to map them to multi-cores. Control dependence happens when a prior thread selects which thread will be speculative and which path will be the mostly likely path for speculation.

B. Dataflow in Thread-Level Speculation

With the support of TLS, dataflow correlated techniques have often been used to realize value prediction [12], branch mis-prediction [13], synchronization and concurrency as well as speculation [14], etc.

Reference [15] exploited the dataflow execution model for a thread-level recovery scheme. The results showed that the redundant execution of dataflow threads could efficiently use underutilized resources in a multi-core. Reference [16] used data-flow principles to realize a scalable thread scheduling co-processor.

With respect to handling the issues of synchronization, concurrency, and speculation, Reference [14] provided a brief overview of dataflow including the concepts, languages, historical architecture, and recent architecture, and indicated that dataflow had inherent advantages in concurrency, synchronization, and speculation over control flow or imperative implementations.

C. Presentation of Early Thread-Level Speculation

Over the past decade, the frequency of commercial processors has no longer increased, but is still subject to Moore′s Law due to the growing number of on-chip processor cores. To handle this issue, TLS has rapidly developed as this technique permits dependence between concurrent threads to make the threads which are difficult to be parallelized speculatively, so an increase of parallelism degree is obtained

This kind of model is realized by hardware, and this model permits threads with data dependence to run speculatively, and uses value prediction to deal with the dependence, and dynamically validates whether or not the predicted values are in accordance with the actual values, if true, this model will submit the results obtained by speculative execution; otherwise speculative threads will be revoked and restarted to ensure the right logic.

The TLS execution model based on hardware thread level speculation(HTLS) is a chip multi-processor (CMP) [17] technique, which converts the serial program to speculative multi-threading programs with the assistance of compiler technique (like pre-computing slice, pointer analysis, program profiler, thread, etc.), and executes (including out-of-order spawning, Multiversion Cache, RAW violation detection, submitting, and validation logic, etc.) on processors with HTLS. This type of automatic parallelization has been verified, and has very limited speedup performance and scalability on account of it for being difficult to control mis-speculation and other overheads.

III. Classification of Thread-Level Speculation

The basic idea behind most TLS techniques is to make full use of multi-core resources to realize the coarse-grained parallelization of serial programs. We implemented classification for TLS, and divided TLS into three types: (1) Hardware Thread-Level Speculation (HTLS); (2) Software Thread-Level Speculation (STLS); and (3) Algorithm Thread-Level Speculation (ATLS). Furthermore, STLS can also be classified into four aspects: (i) Explicit Speculative Parallelization (ESP); (ii) Implicit Speculative Parallelization

(ISP); (iii) Automatic Speculative Parallelization (ASP); and (iv) Software Transactional Memory (STM).

IV. Recent Advancements in TLS

With the rapid development of thread-level speculation, people are now focusing on machine learning-based thread level speculation [8], [9], [18], and algorithm thread level speculation (ATLS) [7], which are gradually becoming a research point to obtain the parallelization effects in development and usage with a more abstract level, and play a fundamental role in overcoming difficulties in programming.

A. Machine Learning-based Thread-Level Speculation

1) Framework of Machine Learning-based Thread Level Speculation: This subsection presents a framework of machine learning-based thread partition, that is illustrated in Figure 2. The framework consists two phases, *i.e.*, training phase (located in the solid box) and validation phase (located in dotted box). The core of framework is the setup of K-Nearest Neighbor (KNN) prediction model (just as an example) and application of it. During the training phase, vector-based features are extracted from training programs as input of KNN learning algorithm, to construct a prediction model. During the validation phase, the already trained prediction model is used to predict the partition schemes for unknown programs. The training programs are different from the final run programs used in validation phase. A cross validation method [19] is used in the collection of hybrid sample set [20] to generate the training sample set.

To predict an appropriate partition scheme for an unseen program, machine learning-based thread level speculation use one machine learning method-KNN to complete the construction of a predictor, that can predict the partition process of it. Once KNN prediction model is trained by training data (programs), it can perform the partition prediction for unknown programs. There are four key points for our approach, including construction of sample set, extraction of features, similarity calculation, prediction model, application of prediction model.

The next critical step is to complete the similarity comparison between the features (present in the form of vectors) generated from unseen program and the ones from training programs. The comparison of them is a searching process within the training samples$'$ space. Once an unseen program comes, the similarity between it and every training program will be calculated to complete the achievement for the most similar training sample.

B. Algorithm Thread Level Speculation

1) Descriptions of Algorithm Thread Level Speculation: The reason why Algorithm Thread-Level Speculation (ATL-S) has gradually become a future point is that many problems existing in irregular programs (shown in Artificial Intelligence, computational biology, finite element analysis, data mining, social network, simulations, N-body problems) [21] still cannot be solved by conventional methods. As conventional methods are seldom related to irregular algorithms, they lack an awareness of parallelization and locality of irregular programs. However, the existing awareness about the biggest obstacle before irregular algorithms is the difficulty in cleaning up all the ambiguous dependences. As seen in Figure 3, the implementation framework primarily includes two parts: the analysis model and analysis results.

The analysis model of ATLS includes three parts: the algorithm paradigm, parameter model, and mapping scheme. With the algorithm paradigm, ATLS is able to convert different kinds of algorithms to a common paradigm, so it is easy to deal with. Then, the metric parameters are searched to build the metric model to evaluate the parallel algorithms. Next, a mapping scheme was built so that different algorithms could be mapped to a fixed model. Once a model was analyzed, we put it into application, as is seen in the lower part of Figure 3. Consequently, we used the ATLS model to evaluate the speedups of the algorithms.

V. Conclusions

This paper reviewed the most representative publications on thread-level speculation. The reviews classified the work on this field into three categories: HTLS, STLS, and ATLS. Some of the most important contributions in each of these categories were analyzed to try to identify the issues that affected the primary thought and characters of each class of TLS.

The researches on TLS were done along time. As a consequence, this review first focused on HTLS. The review revealed that there were several fundamental questions that still remained unanswered many years after they were first identified. HTLS is very complex, and is associated with many parameters including data and control dependence, thread granularities, data correctness, etc. It appears that the proper way to handle these issues in HTLS is to design variable hardware mechanisms to answer one or more questions.

We also reviewed publications on STLS and concluded that high memory and mis-speculation overheads were still the main barriers affecting STLS. It is particularly important to consider the reduction of these overheads in light of the current mechanism to realize the optimization of speculation.

As TLS is applied to larger and more difficult irregular programs, it becomes necessary to design a much faster and practical model that retains the capability of parallelizing irregular programs. This review has presented a novel model (ATLS) capable of realizing the thread-level parallelization in the algorithm level, and a better understanding was reached which should allow us to better utilize them in future.

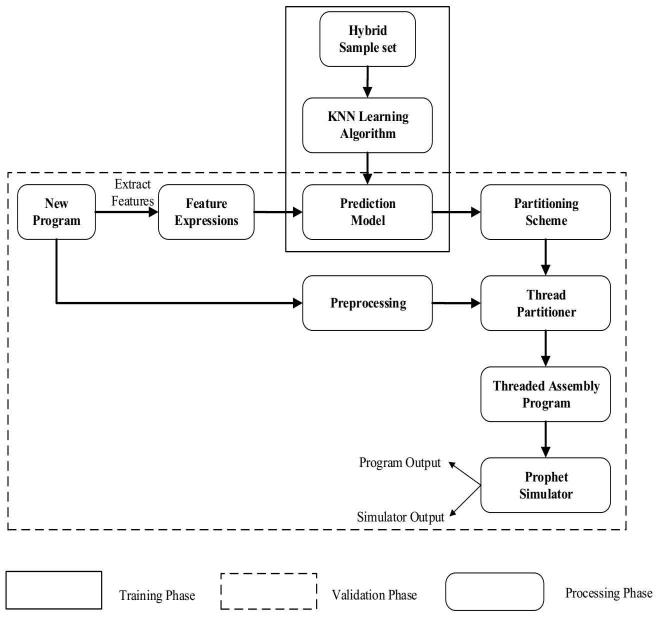

Figure 2: Overall designing framework of machine learning based thread partition

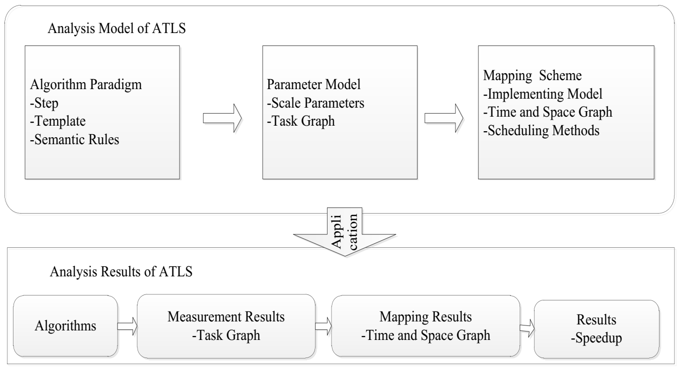

Figure 3: Implementation framework of algorithm thread-level speculation

VI. Acknowledgement

We thank our 505 and 509 laboratory for their great support during our work. We give our best hope to all our colleagues of laboratory for their collaboration. We also thank reviewers for their careful comments and suggestions. This work is supported by National Natural Science Foundation of China Grant No.61772174 and 61370220, and Plan For Scientific Innovation Talent of Henan Province Grant No.174200510011, as well as Program for Innovative Research Team (in Science and Technology) in University of Henan Province Grant No.15IRTSTHN010.

References

[1] Sergio Aldea, Alvaro Estebanez, Diego R. Llanos, and Arturo Gonzalez-Escribano. An openmp extension that supports thread-level speculation. *IEEE Transactions on Parallel Distributed Systems*, 27(1):78–91, 2016.

[2] Alexei Katranov and Alexey Kukanov. Intel; threading building block (intel; tbb) flow graph as a software infrastructure layer for opencl-based computations. In *International Workshop on Opencl*, page 9, 2016.

[3] James Dinan, Pavan Balaji, Darius Buntinas, David Goodell, William Gropp, and Rajeev Thakur. An implementation and evaluation of the mpi 3.0 one-sided communication interface. *Concurrency Computation Practice Experience*, 28(17):4385–4404, 2016.

[4] Richard Gilbert and Srboljub Mijailovich. Distributed multi-scale muscle simulation in a hybrid mpi-cuda computational environment. *Simulation*, 92(1):19–31, 2016.

[5] James Laudon and Daniel Lenoski. The sgi origin: a ccnuma highly scalable server. *Acm Sigarch Computer Architecture News*, 25(2):241–251, 1997.

[6] Henry M. Levy, Susan J. Eggers, and Dean M. Tullsen. Si-

multaneous multithreading: Maximizing on-chip parallelism. 23:392–403, 1995.

[7] Yaobin Wang, Hong An, Zhiqin Liu, Ling Li, and Jun Huang. A flexible chip multiprocessor simulator dedicated for thread level speculation. In *Trustcom/bigdatase/i?spa*, pages 2127–2132, 2017.

[8] Yuxiang Li, Yinliang Zhao, and Qiangsheng Wu. A graph-based thread partition approach in speculative multithreading. In *IEEE International Conference on High PERFORMANCE Computing and Communications; IEEE International Conference on Smart City; IEEE International Conference on Data Science and Systems*, pages 406–413, 2017.

[9] Yuxiang Li, Yinliang Zhao, and Bin Liu. Qinling: A parametric model in speculative multithreading. *Symmetry*, 9(9):180, 2017.

[10] J. G. Steffan, C. B. Colohan, A. Zhai, and T. C. Mowry. Improving value communication for thread-level speculation. 28:65–75, 2002.

[11] Carlos García Quiñones, Carlos Madriles, Jesús Sánchez, Pedro Marcuello, Antonio González, and Dean M Tullsen. Mitosis compiler: an infrastructure for speculative threading based on pre-computation slices. In *ACM Sigplan Notices*, volume 40, pages 269–279. ACM, 2005.

[12] Arthur Perais and Andre Seznec. Bebop: A cost effective predictor infrastructure for superscalar value prediction. In *IEEE International Symposium on High PERFORMANCE Computer Architecture*, pages 13–25, 2015.

[13] A Farcy, O Temam, R Espasa, and T Juan. Dataflow analysis of branch mispredictions and its application to early resolution of branch outcomes. In *ACM/IEEE International Symposium on Microarchitecture, 1998. Micro-31. Proceedings*, pages 59–68, 1998.

[14] Krishna Kavi, Charles Shelor, and Domenico Pace. Chapter twoconcurrency, synchronization, and speculationthe dataflow way. *Advances in Computers*, 96:47–104, 2015.

[15] Sebastian Weis, Arne Garbade, Bernhard Fechner, Avi Mendelson, Roberto Giorgi, and Theo Ungerer. Architectural support for fault tolerance in a teradevice dataflow system. *International Journal of Parallel Programming*, 44(2):208–232, 2016.

[16] R. Giorgi and A. Scionti. A scalable thread scheduling co-processor based on data-flow principles. *Future Generation Computer Systems*, 53(C):100–108, 2015.

[17] Lance Hammond, Benedict A Hubbert, Michael Siu, Manohar K Prabhu, Michael Chen, and K Olukolun. The stanford hydra cmp. *IEEE micro*, 20(2):71–84, 2000.

[18] Yuxiang Li, Yinliang Zhao, and Qiangsheng Wu. Gba: A graphbased thread partition approach in speculative multithreading. *Concurrency Computation Practice Experience*, 29, 2017.

[19] Shuichi Shinmura. The 100-fold cross validation for small sample method. *DATA ANALYTICS 2016*, page 41, 2016.

[20] Yuxiang Li, Yinliang Zhao, and Jiaqiang Shi. A hybrid samples generation approach in speculative multithreading. In *High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2016 IEEE 18th International Conference on*, pages 35–41. IEEE, 2016.

[21] Xiaoqiang Li, Wenting Han, Gu Liu, Hong An, Mu Xu, Wei Zhou, and Qi Li. A speculative hmmer search implementation on gpu. In *IEEE International Parallel and Distributed Processing Symposium Workshops and Phd Forum*, pages 735–741, 2012.