HAL archives-ouvertes.fr

Improving Branch Prediction for Thread Migration on Multi-core Architectures

Tan Zhang, Chaobing Zhou, Libo Huang, Nong Xiao, Sheng Ma

▶ **To cite this version:**

Tan Zhang, Chaobing Zhou, Libo Huang, Nong Xiao, Sheng Ma. Improving Branch Prediction for Thread Migration on Multi-core Architectures. 14th IFIP International Conference on Network and Parallel Computing (NPC), Oct 2017, Hefei, China. pp.87-99, 10.1007/978-3-319-68210-5_8 . hal-01705440

HAL Id: hal-01705440

https://hal.inria.fr/hal-01705440

Submitted on 9 Feb 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution| 4.0 International License

Improving Branch Prediction for Thread Migration on Multi-Core Architectures

Tan Zhang, Chaobing Zhou, Libo Huang*, Nong Xiao,Sheng Ma

School of Computer, National University of Defense Technology
Changsha 410073, China
{zhangtan15, zhouchaobing15, libohuang, nongxiao, masheng}@nudt.edu.cn

Abstract. Thread migration is ubiquitous in multi-core architectures. When a thread migrates to an idle core, the branch information of the branch predictor on the idle core is absent, which will lead to the branch predictor works with comparatively low predicition accuracy until the warm-up finish. During the warm-up period, the branch predictor spends quite a lot of time on recording branch information and often makes mispredictions. These are the main performance impact of thread migration. In this paper, we point out that, when a thread migrates to an idle core, the prediction accuaracy can be improved by migrating branch history information from the source core to the target. Moreover, several migration strategies are introduced to fully exploit the performance of branch prediction migration. Experiment results show that, compared to the experiment baseline which dosen't migrate any branch history information, branch prediction migration reduces MPKI of the branch predictor on new core by 43.46% on average.

Keywords: Branch Prediction, Thread Migration, Multi-core Architecture

1 Introduction

Multi-core processors which bring up large amounts of powerful computational resources, are ubiquitous today in PC and servers. To make better use of the available cores and expose parallelism, thread migration is desired in multi-core processors. In [4], the authors point out that the most challenge work of thread migration is the migration of cache and Branch Predictor (BP). Accurate branch prediction is essential to achieving high performance for processor design. Modern processors employ increasingly complicated branch predictors to sustain instruction fetch bandwidth that is sufficient for wide out-of-order execution cores. It relies large amounts of branch history information for very high prediction accuracy. However, when a thread migrate to an idle core, branch information is absent and the predictor fails to work with high prediction accuracy, which is one of the main impact of thread migration performance. To solve this problem, we migrate branch history information from the source core to the target core to accelerate the warm-up process. With the help of branch

* corresponding author

information migration, the target core dosent have to train the branch predictor from scratch. In addition, the branch accuracy will be improved.

To Migrate branch history information more efficiently, we couldnt simply migrate the whole branch information to the idle core because migrating branch history will occupy the bus between cores and consume instruction cycles. Based on different thread length and branch information length, we should take corresponding strategies. In order to investigate the proper migration strategy in different scenarios, we introduce several simple migration strategies. There are two categories of BP migration including migrating RAM or migrating RAM&GHR(Global History Register). For the situation of migrating RAM, the migration strategies are divided into 3 classes migrating the whole RAM or a part of RAM or only the basic prediction table T0. For migrating GHR, we introduce several simple strategies to modify GHR value. The GHR does hash with PC to index the prediction table. Since the GHRs value will be updating during the migration, the branch predictor will be indexed to the wrong prediction table if we dont modify the GHR value. Based on the discussion above, finding the proper strategies for different situations is the key problem of BP migration. Some exploration work about GHR migration is shown in [5], which points out that the best solution does not transfer any global history but rather synthesizes a value in the new cores GHR. In such a way, it allows each unique thread to update and use its own branch history immediately. Compared to their work, we take migration overhead into consideration. The bus width and the RAM size is known, so we can calcul ate the time cost of migrating branch predictors. They assume that the BP history can be migrated in a negligible short time for short thread. In our work, we take BP migration overhead into consideration and point out that if the thread has less than 1000 instrucitons, migrating BP won't get performance improvement. In addtion, we investigate the impact of the RAM size and the thread length.

This paper is to reduce or eliminate the cost of BP training and increase the accuracy of branch predictor for thread migration by migrating branch predictor information. With extensive simulation experiments, we find that if we only migrate RAM without GHR, instead of migrating all sub-predictors, we can reduce MPKI of the branch predictor on new core by 43.46% on average. This strategy performs steadily and is suitable for many scenarios. The remainder of this paper is organized as follows. Section 2 presents the Background of our work. Migration model is introduced in Section 3. Section 4 presents our experiment methodoloy. Performance evaluation and the discussion of migration strategies are presented in Section5. Section6 presents the related work. Finally, Section7 concludes this study.

2 Background

Accurate branch prediction is essential to achieving high performance for multicore processor design. In this work, we choose a representative branch predictor– TAGE to investigate the BP migration strategies. The TAGE predictor is often

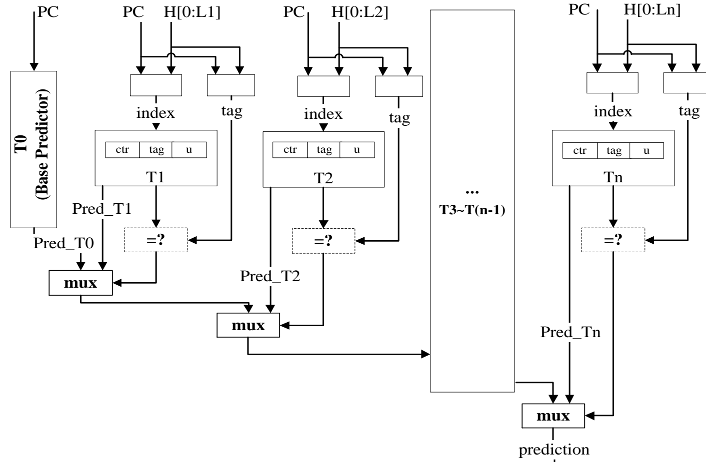

Fig. 1. TAGE Structure

considered as state-of-the-art in conditional branch predictors proposed by academy. It is widespread in multi-cores because the implementation on hardware is easy.

2.1 TAGE Predictor

Structure of TAGE As shown in Fig. 1 , TAGE predictor consists of a base predictor T0 and n (T1,T2$\cdots$Tn) tagged predictor components. T0 is a 2-bit counter bimodal table indexed by the PC. It provides a basic prediction when other predictors are not matching. Ti ($i > 0$) is indexed by a hash result of PC and length L(i) branch history. L(i) is the number of GHR's bits we used to index Ti. And it satisfies the geometric progression, i.e L(i) = (int)($\alpha^{i-1} * L(1) + 0.5$); ($i > 1$). Each tagged component contains a number of configurable entries. The design of TAGE is flexible. If the size of RAM are same, we can get different number of entries with different entry structures. Each entry is composed of ctr, tag and u. Ctr (pred, 3 or 2bit) is a saturation counter, used to indicate the direction of a branch (if $|2 * ctr + 1| \geq ([1 \ll ctrwidth] - 1)$), the prediction is strong, else the prediction is weak). Tag is used to confirm that this entry is matches the branch. The tag width of each component can be different. But in the same component, different entries have the same tag size. U (useful, 2 or 1 bit), which is an age counter, indicates whether the entry is valid.

3 Model

In this section, we present our thread migration model in multi-cores. In order to simplify the BP migration problem, we simply take dual-core architecture into consideration. Each core has a RAM which store branch prediction table of the TAGE predictor. The RAM has a port that can be read and written. We transmit branch prediction information when the port is idle. There is a bus

between two cores. Each core can transmit information to another though the bus.

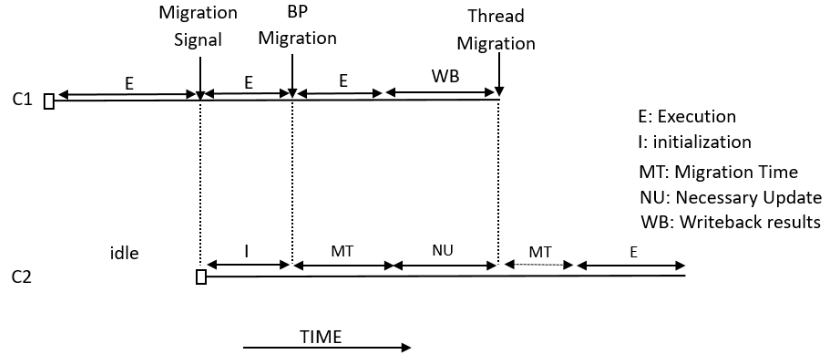

Fig. 2. Thread migration model

3.1 BP Migration Model

Our BP migration models are shown in Fig. 2. During the execution of a thread on a core(We denote as C1), there is a need for migration (e.g, load balance or temperature threshold has been reached), and it is decided that the execution should be transferred to another idle core, C2. As soon as this decision is made, a migration signal is send to the C2. When C2 is activated, it enters first the initialization phase where its resources are turned on. After initialization, the resources that a core running need is prepared. Then we are able to migrate C1's BP to C2. When the initialization finished, C2 enter the migration phase. After the C1 finish executing the remaining instructions in the pipeline, it will acquire the bus. The branch prediction migration will be interrupted and C2 enters the necessary update phase. This is the phase where state essential for correctness is updated. During this phase the architectural register state is transferred from C1 to C2, and the dirty cache blocks, in a private cache to be turned off in C1, are written in a lower level cache. When the NU phase completes, C1 becomes inactive and C2 starts to migrate the remaining branch prediction information. If the RAM on C1 has been altered during the migration, there are two cases. If the prediction table already has been migrated to C2, we keep its previous value. If the prediction table has not been migrated, we read and migrate its new value.

4 Methodology

In this section, we will describe our simulation framework, baseline, and benchmark selection.

4.1 Simulation Setup and Methodologies

We run the target benchmark on GEM5 simulator [7] and Intercept branch prediction data that conform to the CBP2014 [8] trace format specification [9]. Based on the BP simulation environment, we evaluate various BP migration strategies. To obtain simulation experiment trace, we modified the GEM5 code to print out the instructions execution information of our benchmark [8,9]. The simulation experiment trace contains 100 million instruction records. As shown in Table 1, each record contains four fields. The PC is the unique identifier for the execution instruction, known as program counter. OpType indicates the type of the instruction. The BranchTaken field records the actual branch jump information of the PC's corresponding instruction. The BranchTarget is the destination address for the jump. We configure gem5 for the arm v7 instruction set of single-core structure, the specific parameters shown in Table 2.

Table 1. Record in trace

PC	opType	branchTaken	branchTarget

Table 2. System parameters setting

CPU	ARM detailed, ARM V7a, O3, 2GHz 3-wide fetch/decode 6-wide dispatch 8-wide issue/commit 40-entry ROB; 64-entry store buffer
L1I cache	32KB, 2-way, 2MHSRs 1-cycle latency
L1D cache	32KB, 2-way, 6MHSRs 2-cycle latency
L2 cache	Unifined 1MB, 16-way, 16MHSRs 12-cycle latency
Branch predictor	LSC-TAGE

Evaluation Criterion: We use MPKI(Misprediction per Kilo Instructions) as the evaluation criterion. CBP uses a trace-driven simulation. As the simulation begin, CBP gets the instruction record constantly. If the record is a conditional branch instruction, then we use branch prediction and get the prediction value. We update branch predictor with the prediction jump value, the actual jump value, the PC and the jump destination address. If the prediction jump value is inconsistent, then we record the failure times. When the trace is resolved, we calculate MPKI of the entire trace as the output.

The Simulation of the migration process : In order to simulate the process of thread migration, we modified the CBP2014 simulation framework. Firstly, the trace runs on core 1. When the trace runs to the instruction that the thread start to migrate (we call this instruction Migration Start Point -MSP), we start to migrate branch predictor to core 2 with specified bandwidth. Core 1 continues to complete the instructions in the pipeline. When the trace runs

Table 3. parameter setting of TAGE

Number of sub-predictors (N)	11
GHR length(bits)	360
Number of T0's entry(log size)	12
Entry width of T0 (bits)	2
Number of Ti's entry(log size)	8,8,8,8,7,8,7,6,6
tag width of Ti (bits)	7,7,7,8,9,10,10,11,13,13
Entry width of Ti (tag,ctr,u)	10,10,10,11,12,13,13,14,16,16
History length (bits)	4,9,13,24,37,53,91,145,256,359

to the instruction that the migration finish (we call this instruction Migration Over Point -MOP) on core 1, core 2 waits core 1 finish the remaining instructions and returns registers, dirty cache blocks. Then core 1 stop running trace, core 2 continues to run the remaining trace. In the basic version, we assume that IPC is 1 and the bus bandwidth between cores is 128 bits. The RAM size of TAGE branch predictor is 4KB. The specific configuration parameters in Table 3.

5 Evaluation

In this section, we simulate various migration scenarios through a trace-driven approach to evaluate our migration strategies.

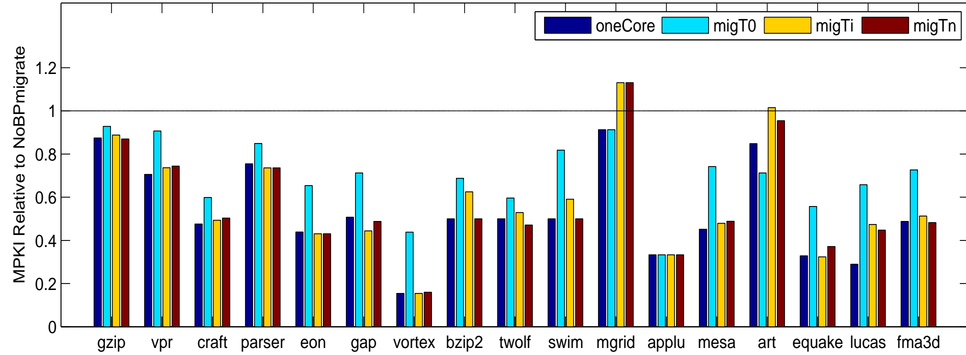

Fig. 3. Migrating RAM only

5.1 Migrate GHR&RAM

Fig. 3 shows the strategies only migrate RAM. The x axis represents different programs. While the y axis means the MPKI of migration strategies divide the MPKI of our baseline. **OneCore** means the trace only runs on a single core. OneCore is the ideal situation and its MPKI should be minimum theoretically. The TAGE branch predictor consists of a basic predictor T0, a GHR and a number of tagged sub-predictors. We define **migT0** as only migrate T0. **migTi** means we migrate T0 and several sub-predictors(T1∼Ti, $0 < i < n$). **migTn** represents we migrate the whole RAM(T0∼Tn). **NoBPMigrate** means thread migration

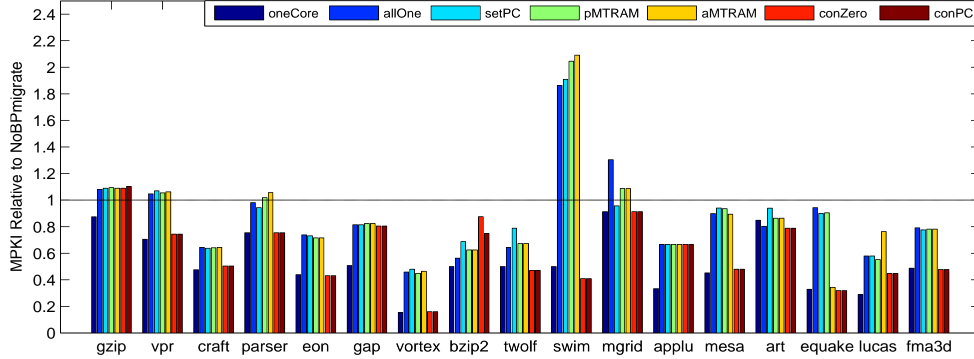

Fig. 4. Migrating RAM and GHR with different strategies

without migrating BP. Comparing each strategy with **NoBPMigrate**, we can get the prediction accuracy improvement of corresponding strategy. If the bar in Fig. 3 under the horizontal line, it means the corresponding migration strategy can improve the prediction accuracy. As we can see in this Figure, migrating the whole RAM has the best prediciton accuracy improvement. Because more branch prediction informations give predictors more reference.

Fig. 4 shows the strategies which migrate RAM together with GHR. Similarly, the x axis represents different programs. While the y axis means the MPKI of migration strategies divide the MPKI of our baseline. **PreMtRAM** means we migrate GHR before migrating RAM. **AftMtRAM** means we migrate GHR after RAM Migration finish. There two strategies represent different GHR migration time strategies. Since the RAM may be updated on core 1, if we migrate GHR before migrating RAM, GHR's high bits might be not matched with RAM. So we should revise GHR's high bits. **ComPC** means we replace high bits with the first instruction's PC on core B. **ComZero** means we set high bits all 0. **AllOne** means we set GHR's every bit 1. The number of high bits depends on a counter, which records how many branch instructions have been executed during migration. Obviously, **conPC** has the best performance. Since GHR and RAM will be update during the migration, to make them match, we should modify GHR's value. But what's the correct modification strategy couldn't be point out by analysis. Because the update of GHR is irregular. This result is only the expexperiment outcomes.

5.2 Different Thread Length

Statistical window represents how many instructions we use to calculate the MPKI. We change this parameter to simulate different thread length. With this method, we are able to find the appropriate strategy for different thread length. We set window size to 5000, 50,000, 100,000 and 1,000,000 instructions. The experiment results are shown in Fig. 5. For the same trace and same strategy, the shortest thread has the smallest MPKI value, while the longest thread has the biggest MPKI value, which means short thread get more performance gains by migrating BP than long thread.

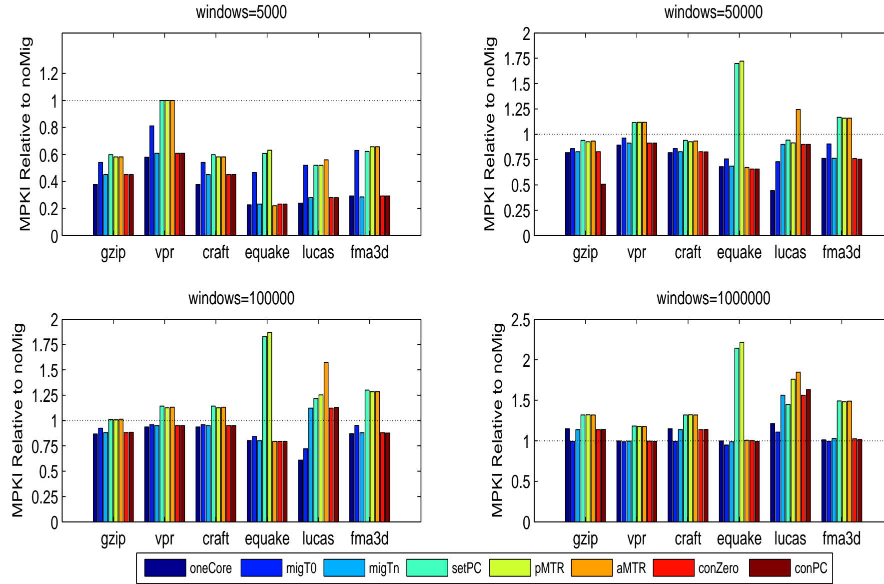

Fig. 5. Different statistical window sizes

Based on the experiment results above, we find that short thread is more suit for BP migration. But if the thread is too short, migrating the whole RAM might cause performance loss for the large migration overhead. We assume each mispredicion will cause 6 instruction cycles loss. Only migrating T0 will have 34 instruction cycles loss, and 83 instrucion cycles cost for migrating T0∼T4. Migrating the whole 4KB RAM will cost 121 instruction cycles. We define the total overhead of misprediction when we do not migrate any BP information as θ. The α_i presents the overhead of i migration strategy. We define the cost of migrating BP as ε and the performance gain of migrating BP as γ. Then we have $\gamma = \theta - \alpha_i - \varepsilon$. Fig.6 shows the performance gain of short thread with different length when the RAM size is 4KB. The x axis shows the thread length and the y axis shows the value of γ for different migration strategies. As we can see in the figure. If the thread is shorter than 1000 instructions, we shouldn't migrate the whole RAM. Only migrate T0 or T0∼T4 has better performance. As the thread length increase, migrating the whole RAM performs better but still worse than migrating Ti. The results show that, for short thread, we migrate part of RAM is the better choice because short thread has relatively few instructions. The performance gain we get by improving prediction accuracy can't cover the cost of migrating whole RAM.

5.3 RAM Size

In order to explore the performance of BP migration under different RAM sizes, we use the DSE algorithm [10] to explore the proper TAGE parameters with RAM sizes of 8KB, 12KB, 16KB, 24KB and 32KB. The windows size is 10,000

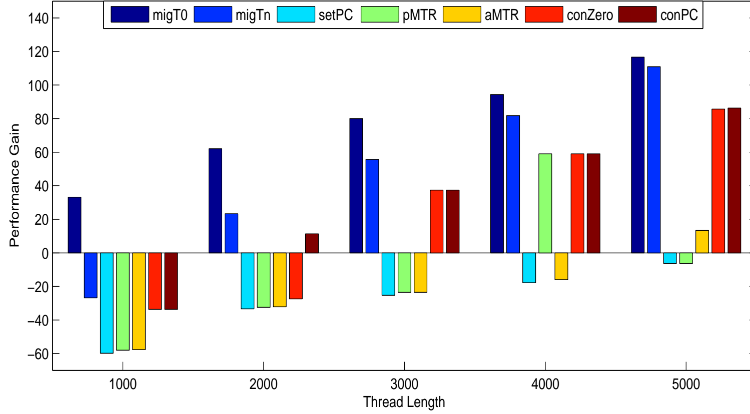

Fig. 6. Different short thread length

instructions, and the MSP is set to the 7,000,000th. Fig.7 shows the experiment results with different RAM sizes. We intuitively think that the MPKI should decrease with the increasement of RAM size for more branch information will improve the accuracy of branch prediction. But experiment results show that the RAM dosen't have any influence on MPKI since there is no law of the curves in Fig.7. As we mentioned in Section 2, TAGE has a base predictor T0 and n sub-predictors(T1∼Tn). Ti is indexed by a hash result of PC and length L(i) branch history. Actually, Ti indexed by short history length is much easier to be matched. Those indexed by long history length like Tn are rarely used. Although we increase the RAM size, which will increase the number of sub-predictors, the increased sub-predictors are rarely used. In addition, it will increase the overhead of migration. In order to verify our assumption. We compare MPKI of migrating T0∼Tn with migrating T0∼T4 and only migrating T0. As we can see in Fig. 8, both migT4 and migTn outperform migT0. But migT4 and migTn have similar performance. So it proves that only increase RAM size is inefficient.

5.4 Discussion

Based on the experiment results, we find that migrating the whole RAM without migrating GHR is the best migration strategy. We set the statistic window size as 100,000 and the RAM size as 4KB. For the thread on the idle core, it reduces MPKI by 43.46% averagely. For short thread, migrating the whole RAM is unadvisable. As we demonstrated in the experiments, migrating T0∼T4 is the best strategy for short thread.

6 Related Work

Thread migration techniques for reducing power-density and load balance were considered in previous work [11] [4, 12, 13]. [12] presents a technique which simultaneously migrates data and threads based on vectors specifying locality and resource usage. This technique improves performance on applications with distinguishable locality and imbalanced resource usage. In striving to exploit the

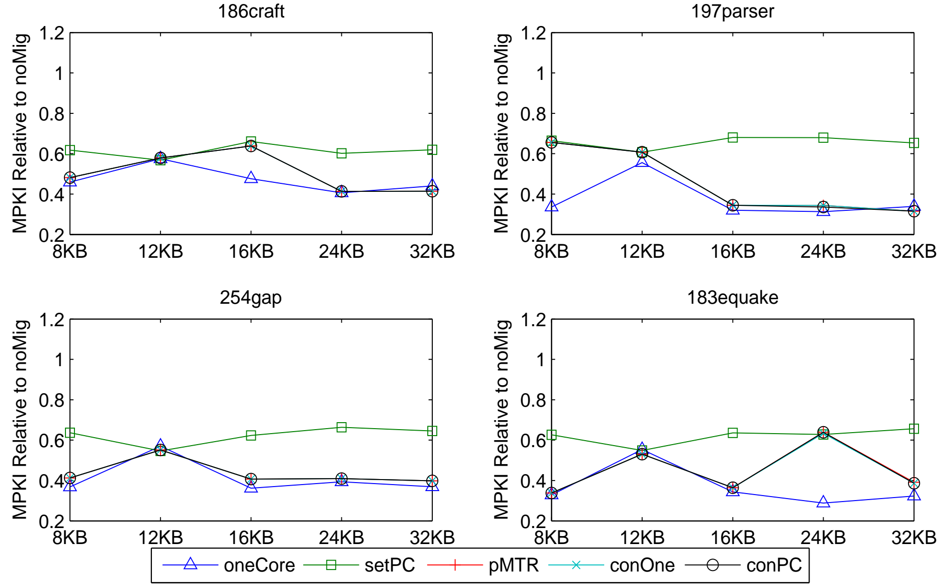

Fig. 7. Different RAM sizes

ever-increasing availability of cores, compilers will employ techniques that create short threads to extract parallelism on multiple cores [14]. These papers try to achieve load balance or exploiting parallelism for multi-cores with thread creation or migration method. However, our purpose is to accelerate the BP training and increase prediction accuracy of Branch predictors.

Nowadays state-of-the-art predictors are derived from two categories of predictors: the neural-inspired predictors [15, 16]and the TAGE-based predictors [9, 17–19]. With equivalent storage budgets, the TAGE predictor that only uses global branch history generally outperforms neural global history predictors [18]. TAGE relies on prediction by partial matching and the use of geometric history lengths for branch history. [19] points out that branch prediction accuracy has relied on associating a main state-of-the-art single scheme branch predictor with specialized side predictors. It develop IUM(Immediate Update Mimicker) predictor approach for TAGE. Recent researches mainly focus on adding various small predictors in TAGE branch predictors to increase the prediction accuracy. [9,16] dedicate a small portion of their storage budget to local history components in addition to a large global history component. Such as, ITTAGE for the indirect branch prediction, COTTAGE combining TAGE and ITTAGE [17], TAGE-SC-L integrating the loop predictor and statistics corrector [9]. The purpose of these articles is to improve the performance of the TAGE branch predictor on a single core. However, this paper tries to improve branch prediction accuracy for multi-core architecture.

Previous work [5] demonstrates that significant branch predictor state is lost when control transfers across cores or contexts in a multiprocessor or multi-threaded architecture. They point out that the critical loss of state lies entirely

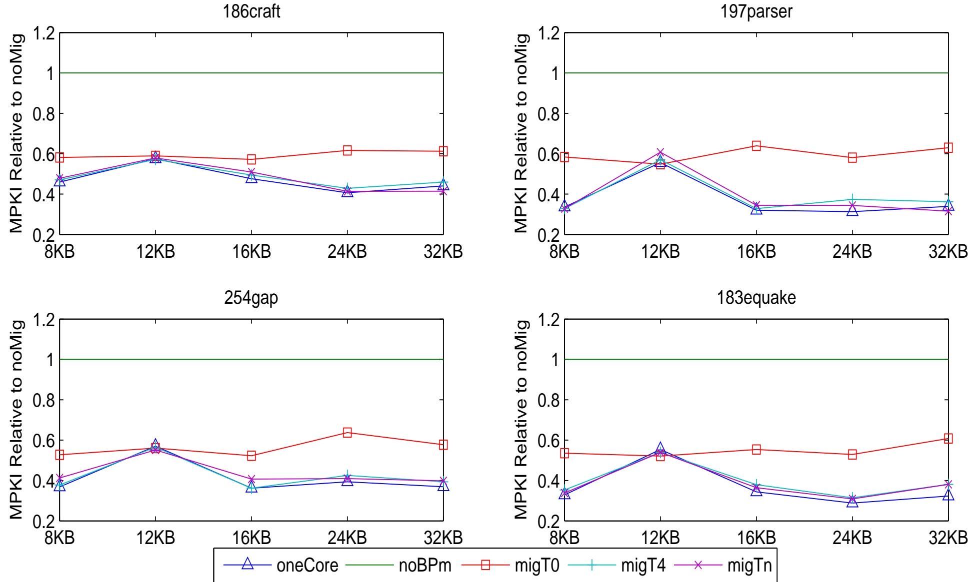

Fig. 8. Migration of T0, T0∼T4 and T0∼Tn

with the global branch history, as stored in the GHR. A thread migration model is given in [4]. The authors propose a training phrase which is to train BP on a new core during the thread migration. Our work's purpose is to accelerate the training phrase and increase BP's accuracy. [20] shows that more than 90% (up to 96%) of instruction cache accesses are common on cores executing a given homogeneous server workload. The authors share instruction execution information between cores using a dedicated storage. Our work is to migrate BP's local history and global history.

7 Conclusion

This paper investigates the performance implications of different BP migration strategies on multi-core Architecture. The main conclusion of our work, is that the branch prediction accuracy can be increased when a thread migrate to execute on an idle core by migrating branch prediction informations. We investigates several different migration strategies with different parameter settings. The simulation experiment results shows that we only migrate RAM can get the best performance for the most scenarios. The results also suggest that if the thread has less than 1000 instrucions, we are not able to get performance gain by migrating BP because of migration overhead. In addition, if the thread is rather short, only migrating part of RAM has the best performance.

8 Acknowledgments

This work is supported by NSFC (No.U1435217, No.61472435, No.61672526 and No. 61433019).

References

1. Shim, K.S., Lis, M., Khan, O., Devadas, S.: Thread migration prediction for distributed shared caches. IEEE Computer Architecture Letters **13**(1) (2014) 53–56
2. Lim, C.H., Daasch, W.R., Cai, G.: A thermal-aware superscalar microprocessor. In: Quality Electronic Design, 2002. Proceedings. International Symposium on. (2002) 517–522
3. Kumar, R., Farkas, K.I., Jouppi, N.P., Ranganathan, P.: Single-isa heterogeneous multi-core architectures: the potential for processor power reduction. In: Ieee/acm International Symposium on Microarchitecture. (2003) 81
4. Constantinou, T., Sazeides, Y., Michaud, P., Fetis, D., Seznec, A.: Performance implications of single thread migration on a chip multi-core. Acm Sigarch Computer Architecture News **33**(4) (2005) 80–91
5. Choi, B., Porter, L., Tullsen, D.M.: Accurate branch prediction for short threads. Acm Sigops Operating Systems Review **42**(2) (2008) 125–134
6. Weissman, B., Gomes, B., Quittek, J., Holtkamp, M.: Efficient fine-grain thread migration with active threads. **17** (1998) 410–414
7. Binkert, N., Beckmann, B., Black, G., Reinhardt, S.K., Saidi, A., Basu, A., Hestness, J., Hower, D.R., Krishna, T., Sardashti, S., et al.: The gem5 simulator. ACM SIGARCH Computer Architecture News **39**(2) (2011) 1–7
8. : CBP2014
9. Seznec, A.: Tage-sc-l branch predictors. JILP - Championship Branch Prediction (2014)
10. Zhou, C., Huang, L., Li, Z., Zhang, T., Dou, Q.: Design space exploration of tage branch predictor with ultra-small ram. In: The 27th edition of the ACM Great Lakes Symposium on VLSI (GLSVLSI), ACM (2017)
11. Chen, J., Shen, L., Wang, Z., Li, N., Xu, Y.: Dynamic power-performance adjustment on clustered multi-threading processors. In: Networking, Architecture and Storage (NAS), 2016 IEEE International Conference on, IEEE (2016) 1–2
12. Shaw, K.A., Dally, W.J.: Migration in single chip multiprocessors. IEEE Computer Architecture Letters **1**(1) (2002) 12–12
13. Lin, F.C.H., Keller, R.M.: The gradient model load balancing method. Software Engineering IEEE Transactions on **SE-13**(1) (1987) 32–38
14. Hum, H.H.J., Maquelin, O., Theobald, K.B., Tian, X., Gao, G.R., Hendren, L.J.: A study of the earth-manna multithreaded system. International Journal of Parallel Programming **24**(4) (1996) 319–348
15. St. Amant, R., Jimenez, D.A., Burger, D.: Low-power, high-performance analog neural branch prediction. In: Ieee/acm International Symposium on Microarchitecture. (2008) 447–458
16. Ishii, Y., Kuroyanagi, K., Sawada, T., Inaba, M., Hiraki, K.: Revisiting local history to improve the fused two-level branch predictor
17. Seznec, A., Michaud, P.: A case for (partially) tagged geometric history length branch prediction. Journal of Instruction Level Parallelism (2006)
18. Seznec, A., Miguel, J.S., Albericio, J.: The inner most loop iteration counter: a new dimension in branch history. In: International Symposium on Microarchitecture. (2015) 347–357
19. Seznec, A.: A new case for the tage branch predictor. In: Ieee/acm International Symposium on Microarchitecture. (2011) 117–127
20. Kaynak, C., Grot, B., Falsafi, B.: Shift: shared history instruction fetch for lean-core server processors. In: Ieee/acm International Symposium on Microarchitecture. (2013) 272–283