http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 12 (8): 1563-1570, 2013
ISSN 1812-5638 / DOI: 10.3923/itj.2013.1563.1570

A Predictive Line Buffer Cache with Key Instruction Trace Predictive Strategy

Cao Xiangrong and Zhang Xiaolin
School of Electronics and Information Engineering, Beijing University of Aeronautics and Astronautics, 100086, Beijing, China

Abstract: As cache constitutes the major part of power dissipation of processor system, energy efficient cache design has attracted much attention. Combined with the characteristics of traditional Predictive Line Buffer (PLB) cache and Trace cache, we define the Key Instruction Trace (KIT) which will make the utilization of instruction trace more effectively. A novel predictive schema-"Key Instruction Trace Predictive Strategy (KITPS)" is also implemented for traditional PLB cache at a modest cost. KITPS improve the hit ratio of line buffer effectively and present advantages in both performance and energy consumption. In the experiments, compared to the traditional PLB cache, the PLB cache with KITPS could provide about 31.5% improvement of performance and 18.9% energy saving.

Key words: Cache, instruction trace, CPI, energy efficiency, predictive strategy

INTRODUCTION

Cache is introduced to alleviate the speed gap between processors and memories. Early researches on cache focus on the improvement of performance. Various methods are applied to reduce the access latency and raise the hit rate of cache (Hennessy and Patterson, 2007). With the development of integrated circuit, embedded processors are widely used in power constraints systems. Efficient power issue has become an important design constraint as well as the performance feature.

Cache is typically implemented as on-chip memory and is relatively large compared to the other parts of processor. The Power dissipation of cache has become increasingly dominant in processor system. To achieve better energy efficiency, several approaches have been proposed to trade performance for energy saving. Hasegawa *et al.* (1995) proposed phased-cache which separated the access process of tag array and data array. Although energy consumption is reduced by avoiding access the data array whose tag wasn't hit, it was punished by serious performance degradation. Filter cache tried to alleviate the performance degradation when implementing the energy efficiency design (Kin *et al.*, 1997, 2000). The tiny filter cache is inserted as the first instruction source of processor and it will consume less energy for each successful instruction fetch. Thus, the hit ratio of filter cache is extremely important. Various prediction techniques were applied to improve the hit ratio in the follow-up researches (Inoue *et al.*, 1999; Chen and Chiang, 2005; Tang *et al.*, 2001; Powell *et al.*, 2001). The basic idea of traditional Predictive Line Buffer cache was the same as the filter cache but limited the size of filter to one cache line size. Line buffer architecture would greatly improve the energy efficiency for each hit operation. But the shortcoming is that the dependence of hit ratio is greatly increased at the same time. In order to make PLB cache be practical (Ali *et al.*, 2005) proposed a fast PLB cache design. He applied a predictive strategy by the branch information and efficiently improved the line buffer hit ratio.

Instruction traces which retired from the pipeline are potentially very beneficial for instruction prediction. Figure 1b illustrates the processor contexts when the code in Fig. 1a is executed. For each iteration, a branch mis-prediction will happen which will lead additional latency. The principle of Trace cache is to improve the instruction delivery efficiency by the history trace to avoid the failed prediction (Rotenberg *et al.*, 1999; Hu *et al.*, 2003; Hossain *et al.*, 2002; Yang and Orailoglu, 2006; Tsai and Chen, 2011; Hu *et al.*, 2002). In Trace cache design, instruction traces are usually stored in a structure similar to a FIFO queue. Limited by the size of the queue, only few recent history traces can be utilized. Meanwhile, the operation circuit for the queue must be simple and tiny because the power and timing is sensitive as cache is on the critical path of processor.

In this study, the PLB cache with KITPS is proposed to improve the lack of traditional PLB cache and Trace cache. Combined with the characteristics of PLB cache, we propose the concept of Key Instruction Trace (KIT) which is the only thing need stored in KITPS. As KIT

Corresponding Author: Cao Xiangrong, School of Electronics and Information Engineering, Beijing University of Aeronautics and Astronautics, 100086, Beijing, China

accounts for a very small part of total instruction traces, in a sense, it effectively compensates for the limitation of the queue size. Meanwhile, with the help of KIT, a simple and effective predictive scheme is implemented for PLB cache at a modest cost. As a result, the PLB cache with KITPS will improve both performance and energy consumption.

DESIGN OF PLB CACHE WITH KITPS

Here, we firstly present the principle of PLB cache with KITPS. Then, we describe how to implement the KITPS design. A new instruction delivery process is proposed to employ the line buffer as the first instruction source. An operation mechanism for the Instruction Trace Table (ITT) which stores the KIT is also required. Several essential registers and circuits for the KITPS are also explained.

Figure 2 shows a piece of program in line buffer and normal cache. Rectangles with numbers represent instructions. Arrows with numbers represent instruction trace. The line buffer holds the recently accessed cache line's context. Table 1 summarizes the effect of traditional PLB cache. If the line buffer is miss, the effect of PLB cache is negative. Obviously, the performance of PLB cache is determined by the line buffer hit ratio.

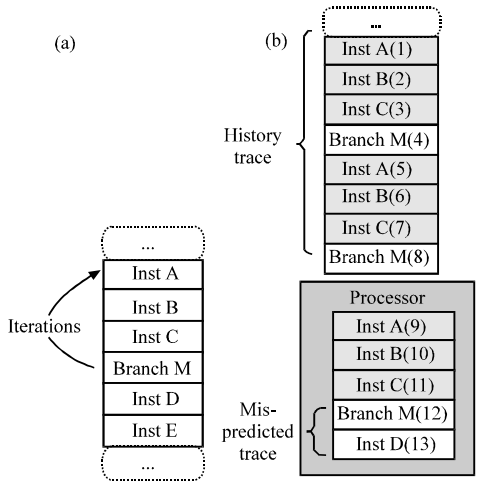

Fig. 1(a-b): Instruction trace of an iterations loop (a) Static code layout and (b) Dynamic instruction trace in a processor

The basic idea of the PLB cache with KITPS is to improve the hit ratio of line buffer as more as possible at a modest cost. We regard the line buffer as the first instruction source. For each fetching instruction, line buffer is visited firstly and only when instruction not found, cache is need to be accessed. Analyzing the program in Fig. 2, line buffer is always hit for trace 1-9 as the instructions are all in the same line. Miss is only caused by some special traces which point to the instruction out of the one line, such as trace 10, 21, 30 and 31. These instruction traces are defined as KIT (red-dotted arrow in Fig. 2).

It strongly prompts us to pay more attention on KIT and if we could do some work to avoid the miss caused by these traces, it will certainly helpful for the line buffer hit ratio. We assume such a mechanism capable of monitoring KIT. When fetching an instruction (e.g., instruction 13) which is just before the miss occurs, it could detect the KIT (trace 10) in advance and update the cache line 1 to the line buffer. Then when we fetch the instruction 18, the line buffer is still hit. The implement of such mechanism will be explained next. This process could improve the line buffer hit ratio without additional latency as it could be carried out synchronously to the operations of traditional PLB cache. From the storage perspective, to record the whole trace in Fig. 2, the size of queue is 31 at least. But in KITPS, only 4 KIT needs stored. In other words, it makes the storage queue much more effective for programs.

Figure 3 shows the logic diagram of PLB cache with KITPS. The predictive logic of KITPS consists of a Last Instruction Address Register (LIAR), ITT, Cache Line Index Register (CLIR) and a line buffer. Line buffer and ITT are visited firstly for each instruction. According to the effective address, if line buffer is hit, return the

Table 1: Traditional PLB cache operations

Line buffer	Action	Effect
Hit	Return Instruction From Line Buffer	Improve
Miss	Same Operations as Normal Cache Access	
	Update the Line Buffer with New Cache Line	Degradation

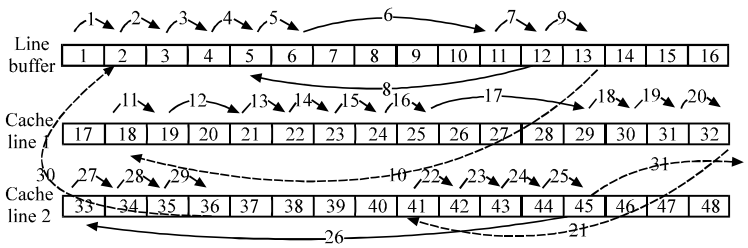

Fig. 2: Instruction trace of program in line buffer and cache (red dotted arrow represents KIT)

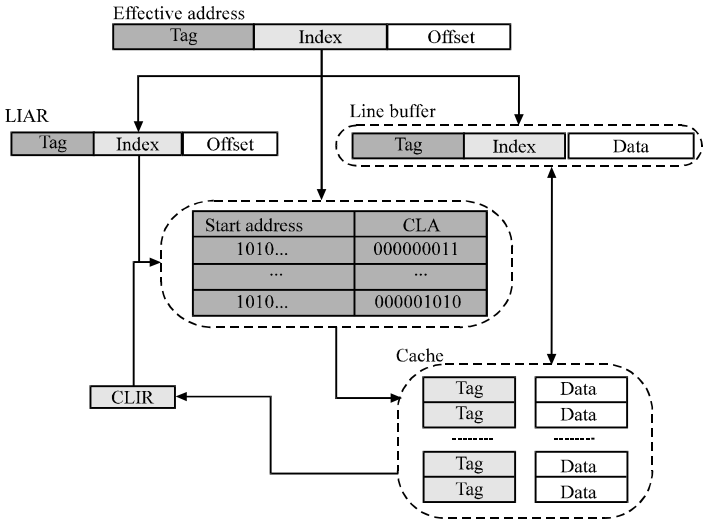

Fig. 3: Architecture of PLB cache with KITPS

Table 2: Operations of PLB Cache with KITPS

Line buffer	ITT	Action
Hit	Miss	Access line buffer and ITT (By comparing the current address to determine whether hit) Fetch instruction in line buffer Return the instruction Update the LIAR with current address
Hit	Hit	Access line buffer and ITT Fetch instruction in line buffer; fetch the CLA in ITT Return the instruction; search the cache line by the CLA Update the line buffer with the cache line found last step; update the LIAR with current address
Miss	Miss	Access line buffer and ITT Access the normal cache Fetch instruction from cache; update the line buffer with the hit cache line Return the instruction; insert a new KIT into ITT(its start instruction from LIAR; its CLA from CLIR and the index of current address) Update the LIAR with current address
Miss	Hit	Acess line buffer and ITT Acess the normal cache Ftch instruction from cache; fetch the CLA in ITT Rtum the instruction; insert a new KIT into ITT; search the cache line by the CLA Udate the line buffer with the cache line found last step; update the LIAR with current address

instruction directly. Otherwise, three following events will happen. (1) Return instruction from normal cache (2) A KIT will be inserted into ITT (3) The line buffer will be updated by the hit cache line in normal cache. If the ITT is hit, it means next trace is a KIT. The line buffer should be updated by the cache line which the KIT points to. Table 2 summarizes the operations of PLB cache with KITPS. The organization of ITT will be explained later.

As shown in Fig. 2, a KIT starts with an instruction and ends with the cache line which including the next target instruction. In KITPS, we describe KIT by the address of the start instruction and a Cache Line Address (CLA) which indicates the location of the target cache line. The LIAR is used to keep the start address of KIT and is updated at the end of each cache access. In the n-way associative cache, to locate a cache line, we need to know which set the cache line belongs to and the cache line index in the set. We can determine the set by the index of effective address and cache line index can directly get from the CLIR. CLA is composed of the index of effective address and CLIR as shown in Fig. 4. For each cache access, only one wordline is chosen (HP Labs and Palo Alto, 2009). The CLIR can be smoothly calculated by a simple encoder with wordlines showing in Fig. 5. The number of bit of CLIR is $\log_2^n$ for n-way associative cache.

As mentioned previously, the ITT search is performed for each cache access. Therefore, the size and structure of ITT present the important issues for the implementation of KITPS. A traditional fully-associative structure, such as the low power CAM design proposed by Efthymiou and Garside (2004) is not suitable for embedded system as consuming significant area and

power. To reduce the overhead of ITT, a 2-way set associative design with a FIFO organization is suggested in Fig. 6. The ITT with size N is divided into N/2 sets physically. When a special set is full, the earliest KIT will be discarded. The effectiveness on hit ratio and energy consumption of ITT of these two organizations are present in Fig. 7. The hit ratios are collected from the simulation platform described in next section and the energy consumption estimations are obtained from CACTI (HP Labs and Palo Alto, 2009). As shown in Fig. 7, the FIFO version has caused an average of 2.25% drop in hit ratio. However, as FIFO has a much more simple circuit, it provides about 74% of energy saving for each access. In our design, we prefer to trade energy saving with slight hit ratio degradation.

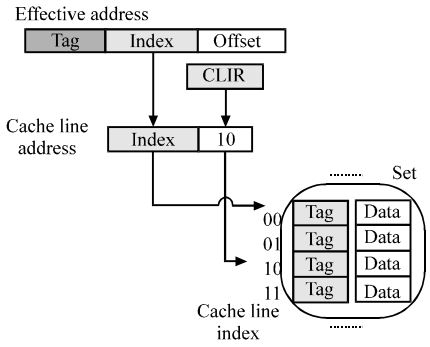

Fig. 4: Cache line address of 4-way associative cache

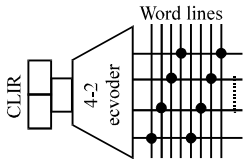

Fig. 5: CLIR circuit of 4-way associative cache

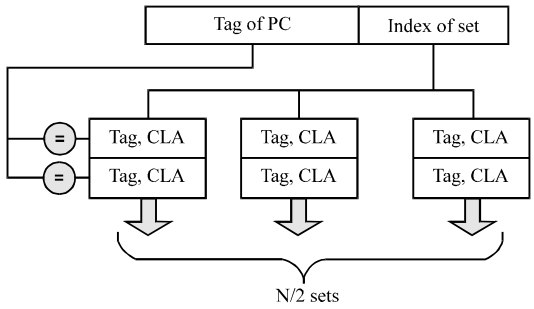

Fig. 6: 2-Way ITT with a FIFO replace policy

EXPERIMENTAL RESULTS

In this section, we present the experimental methodology and the results. To demonstrate the feasibility of PLB cache with KITPS described earlier 2, the design is implemented on a open source CPU "Leon3" which is released by European Space Agency (Gaisler, 2002). The RTL code of modified Leon3 core is synthesized by the Synopsys Design Compiler along with SMIC 0.18 um technology library. Note that only control logic of KITPS is included and the storage unit of general cache is excluded from the synthesis process. Table 3 shows the synthesis results of the modified processor core and the comparisons with the original design. Only 2.5% of cell area and 5.2% of dynamic power is increased for KITPS which is negligible for the overall system.

To reveal the impact on the performance and power of cache during the processor running, we implement the PLB cache on Simplescalar platform for the simulation of a cycle-accurate detailed operation (Austin *et al*., 2002). Referring to the idea proposed by Tsai and Chen (2011), the power model for KITPS is shown in following:

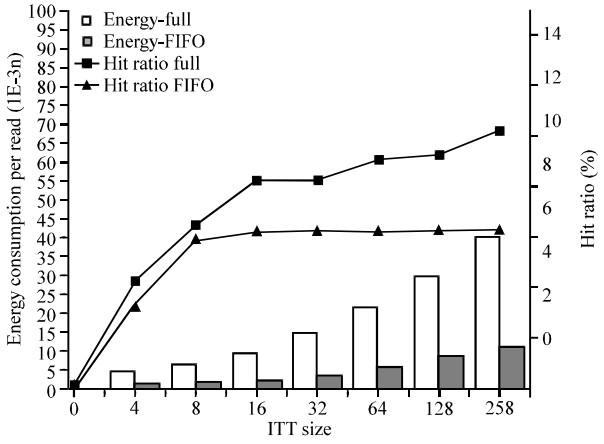

Fig. 7: Energy and hit ratio of full associative and 2-way associative with FIFO

$$E_{dynamic} = N_{read} \times E_{read\text{-}per\text{-}port} + N_{write} \times E_{write\text{-}per\text{-}port} \quad (1)$$

$$E_{static} = P_{leakage} \times Cycle \quad (2)$$

$$E_{module} = E_{dynamic} + E_{static} \quad (3)$$

$$E_{total} = \sum E_{module} \quad (4)$$

The $E_{read\text{-}per\text{-}port}$, $E_{write\text{-}per\text{-}port}$ and $P_{leakage}$ stand for read dynamic energy per port and write dynamic energy per port and leakage energy per cycle and are estimated by the CACTI (HP Labs and Palo Alto, 2009). The N_{read} and N_{write} stand for the number of read access and the number of write access of the main modules of KITPS respectively. These two parameters reflect the impact on energy consumption caused by different access behavior. The Cycle stands for the total cycles of the program executed. These program-related can be obtained by the Simplescalar platform. Each E_{module} is composed of $E_{dynamic}$ and E_{static}, respectively. The E_{total} is derived by summing up all module's energy consumption in (4). For KITPS, the modules include ITT, Line buffer, LIAR, CLAR and normal cache as described in previous section.

Table 3: Synthesis results of leon3 core

	Original	Modified (ratio%)
Cell Area	456325	467732 (102.5)
Power	6.82 mW	7.17 mW (105.2)

We choose 15 benchmarks from SPEC CPU 2006 and MediaBench as the test programs and do the comparative experiments with the traditional PLB cache, Fast PLB cache and PLB cache with KITPS. Table 4 lists the main configuration for the baseline. All benchmarks are running 50 million instructions at most. Without special declare, the ITT size is 8 for KITPS and the line buffer size is 64 byte for all PLB cache in the experiment.

As analysis previously, only when the line buffer is hit, the PLB cache will play an active role. With the help of the simulation platform, it is convenient to monitor the line buffer hit ratio of the three PLB cache. The result is depicted in Fig. 8. The PLB cache with KITPS has the most outstanding performance in the line buffer hit ratio for all benchmarks with about 23.5% improvement compared with the traditional PLB cache. The improvement of Fast PLB cache is about 18%. This proves KITPS is effective. The result mentioned in the experiment is the average of the all 15 benchmarks.

To evaluate the performance of cache, Cycle Per Instruction (CPI) is introduced. The CPI and the CPI Improvement Percentage (CPIIP) compared to traditional PLB cache are shown in Fig. 9. As expected, although cycle punishment will occur when line buffer is miss, CPI is still improved thanks to the help with predictive strategy, 21% improvement of Fast PLB cache and 31.5% improvement of KITPS. In most case, KITPS is better than

Table 4: Base line configuration

Parameters			
Main frequency (MHZ)	200	Level-1 I-cache	16KB, 64-byte cache line, 4-way
Voltage (V)	3.3	Level-1 D-cache	16KB, 64-byte cache line, 4-way
Process (um)	SMIC 0.18	Level-2 cache	None
Temperature (K)	310		

(*cache line size for I-cache should stay the same with the line buffer)

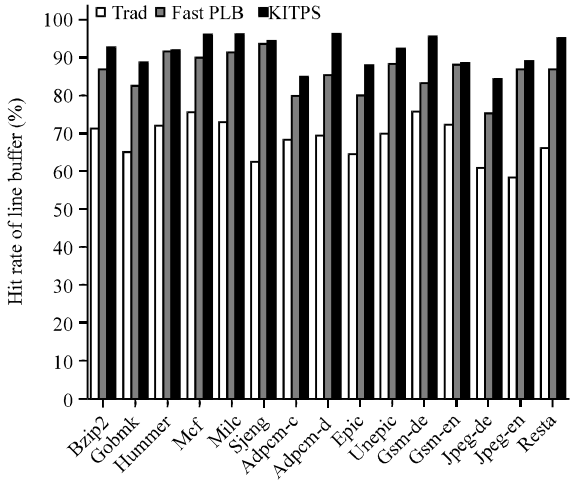

Fig. 8: Line buffer hit ratio of Trad/Fast/ KITPS PLB cache

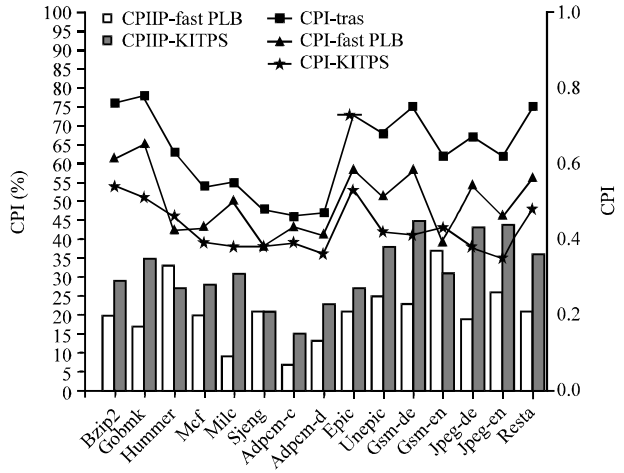

Fig. 9: CPI of trad/fast/KITPS PLB cache

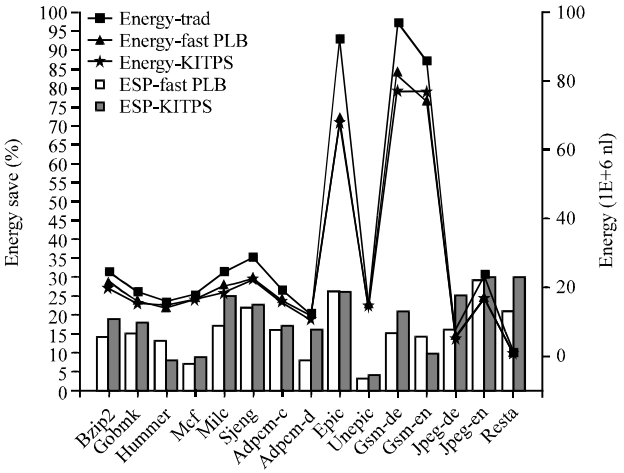

Fig. 10: Energy of trad/fast/KITPS PLB cache

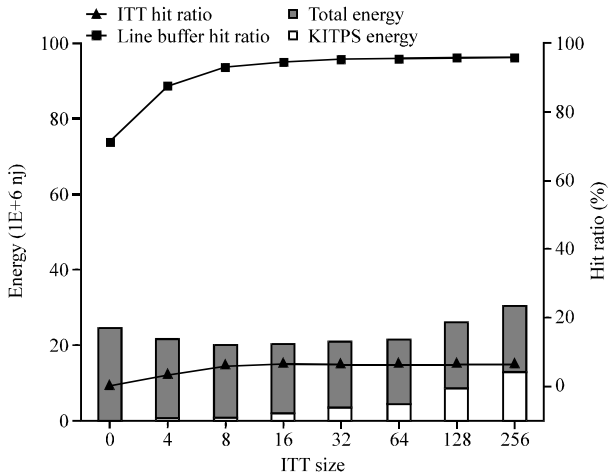

Fig. 11: ITT size effect on energy and hit ratio

Fast PLB except for hummer, sjeng and gsm-en. It indicates that the effect of predictive strategy also relative to the benchmark self.

Figure 10 shows the energy consumption in the experiment. Similar to the CPI case, when line buffer is hit, energy consumption is much less than other cases. Benefit from the improvement of line buffer hit ratio, Fast PLB cache reduces the energy consumption about 15.8% and KITPS reduces the energy consumption about 18.9%. We notice that, comparing KITPS and Fast PLB, the improvement of energy is not notable as the same as the CPI. This could be explained that the predictive circuit for KITPS is more complex which will weaken the energy performance.

ITT size is an important parameter for KITPS. Larger size of ITT means more KIT could be monitored and will apparently improve the hit ratio of line buffer more. The side effect is that it will also cause more energy consumption. Figure 11 depicts the experiment results of different ITT size from 0-256 for benchmark bzip2. Zero size means the traditional PLB cache.

KITPS energy caused by the KITPS logic is a part of total energy in Fig. 11. At the beginning of increasing ITT size, the improvement of hit ratio is obvious. But there is nearly no change for hit ratio after ITT size larger than 16 for bzip2. The KITPS energy is linearly increased with the ITT size and finally leads to the total energy exceeding the traditional PLB cache. It means there is a optimize ITT size for each special program. According to our experiments, the recommended ITT size is between 4 and 32. Supplement, the exact optimize ITT size for a program can only be obtained by experiment until now.

CONCLUSIONS

In this study, the PLB cache with KITPS is proposed to improve both performance and energy consumption of cache. The KITPS consists of an ITT to store the KIT and two additional registers LIAR and CLIR to generate the CLA. By comparing the address of the incoming instruction with ITT, we can detect the KIT in advance and update the right cache line context to the line buffer ahead to avoid the miss of line buffer. The experiment results prove that the KITPS works effectively with 31.5% improvement of CPI and 18.9% saving of energy compared to traditional PLB cache. We also discuss the effect of ITT size and conclude a recommended size range between 4 and 32. A dynamic scheme to obtain the optimize ITT size for a special program will be a valuable issue to research in next work in the future.

ACKNOWLEDGMENT

This study is partially supported by Beijing Key Discipline Foundation (No. XK100060525). The authors also gratefully acknowledge the helpful comments and suggestions of the reviewers which have improved the presentation.

REFERENCES

Ali, K., M. Aboelaze and S. Datta, 2005. Predictive line buffer: A fast, energy efficient cache architecture. Proceedings of the IEEE SoutheastCon, 2006, March 31-April 2, 2005, Memphis, TN., USA., pp: 291-295.

Austin, T.D., E. Larson and D. Ernst, 2002. SimpleScalar: An infrastructure for computer system modeling. Computer, 35: 59-67.

Chen, H.C. and J.S. Chiang, 2005. Low-power way-predicting cache using valid-bit pre-decision for parallel architectures. Proceedings of the 19th International Conference on Advanced Information Networking and Applications, Vol. 2, March 28-30, 2005, Taipei, Taiwan, pp: 203-206.

Efthymiou, A. and J.D. Garside, 2004. A CAM with mixed serial-parallel comparison for use in low energy caches. IEEE Trans. Very Large Scale Integr. (VLSI) Syst., 12: 325-329.

Gaisler, J., 2002. A portable and fault-tolerant microprocessor based on the SPARC V8 architecture. Proceedings of the International Conference on Dependable Systems and Networks, June 23-26, 2002, Bethesda, MD., USA., pp: 409-415.

HP Labs and Palo Alto, 2009. CACTI: An integrated cache and memory access time, cycle time, area, leakage and dynamic power model. http://www.hpl.hp.com/research/cacti/.

Hasegawa, A., L. Kawasaki, K. Yamada, S.I. Yoshioka, S. Kawasaki and P. Biswas, 1995. SH3: High code density, low power. IEEE Micro, 15: 11-19.

Hennessy, J.L. and D.A. Patterson, 2007. Computer Architecture: A Quantitative Approach. 4th Edn., Morgan Kaufmann, San Francisco, CA., USA., ISBN-13: 9780080475028, pp: 198-214.

Hossain, A., D.J. Pease, J.S. Burns and N. Parveen, 2002. Trace cache performance parameters. Proceedings of the IEEE 20th International Conference on Computer Design: VLSI in Computers and Processors, September 16-18, 2002, Freiburg, Germany, pp: 348-355.

Hu, J.S., M.J. Irwin, N. Vijaykrishnan and M. Kandemir, 2002. Selective trace cache: A low power and high performance fetch mechanism. Department of Computer Science and Engineering, College of Engineering, Pennsylvania State University, Philadelphia, USA., pp: 1-18.

Hu, J.S., N. Vijaykrishnan,, M.J. Irwin and M.T. Kandemir, 2003. Using dynamic branch behavior for power-efficient instruction fetch. Proceedings of the IEEE Computer Society Annual Symposium on VLSI, February 20-21, 2003, Tampa, FL., USA., pp: 127-132.

Inoue, K., T. Ishihara and K. Murakami, 1999. Way-predicting set-associative cache for high performance and low energy consumption. Proceedings of the International Symposium on Low Power Electronics and Design, August 16-17, 1999, San Diego, CA., USA., pp: 273-275.

Kin, J., M. Gupta and W.H. Mangione-Smith, 1997. The filter cache: An energy efficient memory structure. Proceedings of the 30th Annual IEEE/ACM International Symposium on Microarchitecture, December 1-3, 1997, Research Triangle Park, NC., USA., pp: 184-193.

Kin, J., M. Gupta and W.H. Magione-Simth, 2000. Filtering memory references to increase energy efficiency. IEEE Trans. Comput., 49: 1-15.

Powell, M.D., A. Agarwal, T.N. Vijaykumar, B. Falsafi and K. Roy, 2001. Reducing set-associative cache energy via way-prediction and selective direct-mapping. Proceedings of the 34th ACM/IEEE International Symposium on Microarchitecture, December 1-5, 2001, Austin, TX., USA., pp: 54-65.

Rotenberg, E., S. Bennett and J.E. Smith, 1999. A trace cache microarchitecture and evaluation. IEEE Trans. Comput., 48: 111-120.

Tang, W., R.K. Gupta and A. Nicolau, 2001. Design of a predictive filter cache for energy savings in high performance processor architectures. Proceedings of the International Conference on Computer Design, September 23-26, 2001, Austin, TX., USA., pp: 68-73.

Tsai, Y.Y. and C.H. Chen, 2011. Energy-efficient trace reuse cache for embedded processors. IEEE Trans. Very Large Scale Integr. (VLSI) Syst., 19: 1681-1694.

Yang, C. and A. Orailoglu, 2006. Power-efficient instruction delivery through trace reuse. Proceedings of the 15th International Conference on Parallel Architectures and Compilation Techniques, September 16-20, 2006, Seattle, WA., USA., pp: 192-201.