TELKOMNIKA, Vol.11, No.11, November 2013, pp. 6454~6459
e-ISSN: 2087-278X

Instruction Pipeline Efficient Mechanism with Maximum Hit Ratio

Shahnawaz Talpur*[1,2], Yizhuo Wang[1], Shahnawaz Farhan Khahro[1], XiaoJun Wang[1], Xu Chen[1], Feng Shi[1]
[1]Beijing Institute of Technology Beijing, China
[2]Mehran University of Engineering and Technology , Jamshoro, Sindh
Corresponding authors, e-mail: talpur@bit.edu.cn, frankwyz@126.com

Abstract

To achieve highest performance in rapidly growing advancement in multi-core technology, there is need to minimize the large gap between faster processor speed and memory. It becomes more critical issue when branch occurs with penalty of cache miss. Many researchers proposed different branch prediction, instruction perfecting methods and algorithms but the CPU pipeline performance couldn't be the maximal. A prototype model has been designed in this paper which has no prediction for branch and no chance of CPU core to be idle. Analysisis carried out on the benchmarks suite and Transactional Slice (TS) has been proposed in contrast with traditional delay slot and dynamic prediction fetch branch. In proposed mechanism hit rate will be maximal. Pin Tool is used to analyze the Transactional Slice with SPEC 2006 benchmark.

Keywords: *conditional branch, branch misprediction, performance evaluation, transactional slice, pipeline processing*

1. Introduction

Currently there is a huge discussion carried out on the processor speed with cache synchronization. Keeping the execution core full of activity, there must be efficient techniques required for instruction cache performance. It is necessarythatinstruction blocks must be perfected in instruction cache, prior the core request, to avoid the processor to be idle.

Many processors have high clock cycle's frequencies leading to longer pipe line. Pentium 4 has 20 and IBM Power has 14 stages [1, 2]. Due to longer size pipeline instruction takes more time to reach execution stage because branches take longer to resolve. So the branch misprediction is morecritical issue in those processors. On the other hand few researchers have the opposite concept that if the pipeline will be deeper,with large enough on-chip cache, the processor frequency will be increase to 100%.

It is observed that majority of application instructions references depend on predictions.Processor resources are utilized on branch predictions in high performance computing. Branches are categorized in static branches may cause 20% mispredictionsand 63% mispredictions may be caused by dynamic branches [3]. Conditional branch prediction is common from the decades in high performance processors [4, 5]. It is also past practice to execute instructions speculatively. There is less chance of incorrect branch prediction and resources available to speculative instructions [6, 7, 8]. For some special applicationsconfidence hybrid branch predicators' mechanism were introduced on prediction histories [9, 10].

Unfortunately, due to the unpredictable nature of code and data streams,the pipeline cannot always be filled correctly and theflushing of the pipeline exposes the latency. For resolving the above issues different effective fetching techniques have beenintroduced. Prefetch aware cache management, SHIP hit predictor on last level cache policies introduces to measure the pre-fetch policies dynamically and avoid cache pollution [11, 12, 13, 14, 15]. Stream prefetcher for mid-level cache used in Intel Core i7 processor [16]. Next-line instruction prefetcher [17, 18], correlating prefetchers [19, 20], allowing the branch predictor to run ahead of instruction fetch. Even though with all these counter measures L1 caches miss rate is over 40% of the execution time [21, 22]. So the researchers have proposed different mechanism to

Received March 19, 2013; Revised July 1, 2013; Accepted July 15, 2013

build a bridge between L1 instruction cache size and the necessity of low-latency access to instructions.

The paper is organized in different sections. Section 2 elaborates the branch misprediction ration comparing the performance with pipeline stages. Section 3 describes our analytical approach of Transactional Slice. Section 4 contains relative work. System configuration of our experimental machine is detailed in section 5. In section 6 we have concluded the paper.

2. Pipeline Branch Misprediction Ration

In this section the useful duration of pipeline with its stagesis analyzed and assesed. Here branchmisprediction algorithm proposed in [23] is referred;

$$B_m = N_s * (cycle\ Time\ -\ Overhead\ Time) \quad (1)$$

Where B_m is Branch misprediction, N_s is No. of stages.

Analysis is done on 2.66 GHz system having 375ps cycle time and has 14 stages per core.Branch misprediction factor is computed with different constantoverheadworkloads as described in Table1. Instruction pipeline is completely filled up for the maximum Transactional Slice length to satisfy our model.

$$Useful\ Time = cycle\ Time\ -\ Overhead\ Time \quad (2)$$

$$Useful\ Time\ =\ 375\ -\ Const.\ Overhead \quad (3)$$

$$B_m = N_s * Useful\ Time \quad (4)$$

Table 1. Performance Ration with Branch Misprediction

Performance (%)	Const. overhead
89.33	40
84.00	60
78.67	80
76.00	90

3. Transactional Slice Behavior

The Transactional Slice is block of instructions which must end with branch instruction. Figure 1 shows the block diagram of TS to pipeline where each TS is in series with pipeline. In traditional system prefetch proceeds to fetch contiguous blocks in memory until a branch predicted as taken reachesthe fetch unit. Branch misprediction and prefetch overhead are two performance losses in front end CPI. After branch occurs there are several mechanism for maximum hit rate in pipeline. For example if the instruction stream has encountered a conditional branch then the CPU cannot know whether the next instruction is the one following the branch or the instruction at the target location until it has evaluated the branch, resulting in a bubble in the pipeline. Hence to handle such a situation some RISC architectures have a branch delay slot, wherein the instruction after the branch will always be executed, no matter whether the branch is opted or not, to improve the efficiency of pipeline.In OOO processor delay slot doesn’t exist so if branch is mispredicted then wait for pipeline to get empty is big loss. On a deeply pipelined processor this would often take longer than the typical number of instructions between branches, so one can be completely stalled. The other mechanism is dynamically predicted branch; predictor can runahead of the instruction cache fetch. The blocks are fetched by branch predictor, put into the prefetch queue and then accessed either from fetch target buffers or branch target buffers. However execution performanceis strictly limited by fetch performance. In our mechanism of Transactional Slice instructions are inserted dynamically in front of the branch instruction. In this technique there will be enough instructions before the conditional branch.

3.1. Count and Trace Mechanism for Transactional Slice

Intel PIN TOOL is used to calulate the Transactional Slice using different workloads of SPEC 2006 benchmarks in this manuscript. The exiting builtin instruction count and instruction trace tools are modified. In this process first simple C++ and C programs are tested to verify the tools. Different types of instruction as well as traces of the branches are analyzed. After verifying on simple applications this Tool is run on SPEC 2006 suite. The experiment results of cpu2006 INT benchmarks are shown in Figure 2 and FLOAT benchmarks are shown in Figure 3, ref data is used for both benchmarks. There is utility in Intell Pin Tool to sychronize with different benchmarks. We configure the config file of cpu2006 accroding our system. First environment variable were set and then run the tool directoly from benchmark prompt. The simulation results show the number of Transcational slice and the number of instructions in each Transactional Slice.

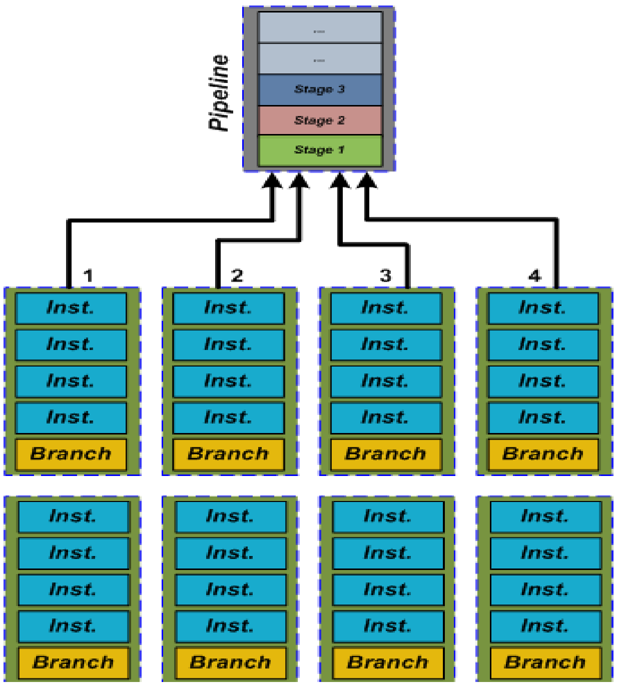

Figure 1. Instructions to Pipeline

3.2. Cycles per Transactional Slice

The formula for calculating the CPU time given in Equation (5) will be used further to calculate Cycle per instruction in benchmark workload.

$$CPU\ Time = IC * CPI * clk\ Cycle\ Time \quad (5)$$

Again the Intell PIN TOOL is used to calculate the number of instructions dynamically in each benchmark workload with same system configuration. Number of instruction were calculated with builtin Tool.

For proposed Transactional Slice the number of cycles will be calculated using above general formula. CPU time to execute one benchmark workload EWT (Execution Workload Time) can be calculated using equation 6.

$$EWT = NIW * CPIW * clk\ Cycle\ Time \quad (6)$$

$$CPIW = \frac{EWT}{NIW * clk\ Cycle\ Time} \quad (7)$$

Where NIW (Number of Instructions in Workload), CPIW (Cycle Per Instruction Workload), CPTS (Cycles per Transactional Slice), STS (Size of Transactional Slice)

Cycle per Instruction of one benchmark workload can be calculated form Equation 7. The SPEC 2006 benchmarks CPIW are calculated using the same formula and results are given in Table 2(a) for C benchmarks and in 2(b) C++ benchmarks. This CPIW can be used to calculate Cycle perTransactional Slice.

$$CPTS = S_{TS} * CPIW \quad (8)$$

Or for calculating the Size we can write the above equation as:

$$S_{TS} = \frac{CPTS}{CPIW} \quad (9)$$

The TS time per instruction in Pipeline (TSTPI) can be determined using Equation 10.

$$TSTPI = clk\ Cycle\ Time * CPIW \quad (10)$$

So the Transactional Slice sizewill be equal to the product of number of pipeline stages and Time of Transactional Slice per instruction.

$$S_{TS} = N_S * TSTPI \quad (11)$$

With this equation we may find in future instruction cache size and the maximum Pipeline utilization.

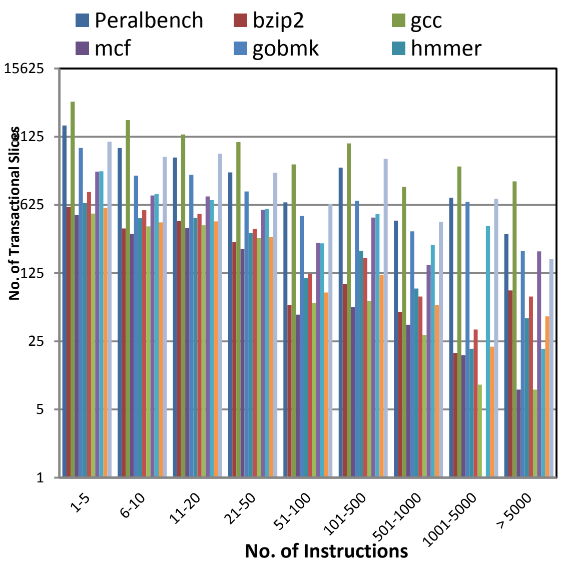

Figure 2. Trasactional Slices in SPEC 2006 INT Benchmarks

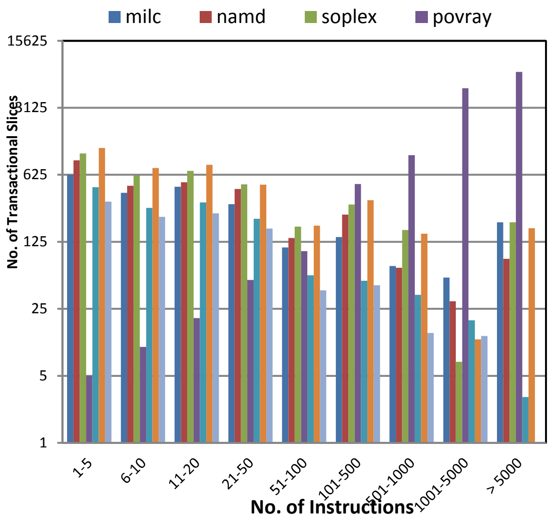

Figure 3. Transactional Slices in SPEC 2006 FLOAT Benchmarks

Table 2(a). C Benchmarks

Benchmark	CPI	Benchmark	CPI
400.perlbench	1.30	458.sjeng	0.88
401.bzip2	5.88	462.libquantum	1.17
403.gcc	11.27	464.h264ref	10.32
429.mcf	4.67	433.milc	1.67
445.gobmk	5.40	470.lbm	2.19
456.hmmer	1.68	482.sphinx3	0.92

Table 2(b). C++ Benchmarks

Benchmark	CPI	Benchmark	CPI
471.omnetpp	7.26	444.namd	7.40
473.astar	2.25	450.soplex	4.15
483.xalancbmk	1.66	453.povray	0.85

4. Related Work

Authors in [24] emphasis on elimination of branch mispredictions which arecaused by slow predecessors as compared to faster microarchitecture core and proposed CFD for separable branches. Authors in [25] described Proactive Instruction fetching technique that records exact sequence number. It implements stall less Fetch-instruction pre-fetcher to improve the performance of L1 instruction cache, avoids the instability and randomness ofthe instruction sequence introduced by the microarchitecture. GPU conditional branch handling mechanism with divergent paths is proposed in [26, 23] measure the performance of processor in terms if branch misprediction latency and fastest branch recovery. Authors in [27] explore the causes of performance loss due to branchmispredictions. They separate it into different categories, e.g. Serialization, window-fill penalty and the Pipeline-fill penalty which is focus of our work. The reason of measuring the size and CPIW of Transactional Slice is to fill up core-pipe line at the peak value to avoid the core to be ideal.

5. System Configuration

We have used Dell machine with linux operating system detailed in below table.

Table 3. System Configuration

Parameters	Description
CPU	Intel(R) Core(TM)2 Quad CPU Q8400 @ 2.66GHz
vendor:	Intel Corp.
size:	2666MHz
width:	64 bits
clock:	1333MHz
cores	4
enabledcores	4
threads	4
L1 cache	
size:	256KiB
capacity:	256KiB
memory	
size:	4GiB
OSgcc version 4.6.3	
x86_64 GNU/Linux	Ubuntu 12.04.2 LTS

6. Conclusion

In this paper an efficient instruction pipeline mechanism is described on the analysis of branch misprediction to get maximum hit ratio. It is observed during this analysis that it has high impact on processor performance vs cache size. The idea of Transactional Slice is proposed in this manuscript, which fill up the pipeline on maximum level. Proposed model will make agreeable change in processor pipeline line and cache synchronization to improve 100% performance.

Acknowledgment

This work is partially supported by National Nature Science Foundation of China under grant No. 60973010.

References

[1] PN Glaskowsky. Pentium 4 (Partially) Previewed. *Microprocessor Report*. 2000; 14(8): 1,11-13.
[2] K Krewell. IBM’s Power4 Unveiling Continues*. Microprocessor Report*. 2000; 1-4.
[3] Jacobsen E, Rotenberg E, Smith JE. *Assigning confidence to conditional branch predictions*. In proceedings of the 29th annual ACM/IEEE symposium on Microarchitecture. 1996; 142-152.
[4] Lee JK, Smith AJ. Branch Prediction Strategies and Branch Target Buffer Design. *Computer.* 1984; (1): 6-22.
[5] Pan ST, So K, Rahmeh JT. Improving the Accuracy of Dynamic Branch Prediction Using Branch Correlation. ACM SIGPLAN Notices. 1992; 27(9): 76-84.
[6] Linley Gwennap. MIPS R10000 Uses Decoupled Architecture. *Microprocessor Report*. 1994; (8): 18-22.
[7] Michael Slater. AMD’s K5 Designed to Outrun Pentium. *Microprocessor Report*. 1994; (8)1: 6-11.
[8] McFarlin DS, Tucker C, Zilles C. *Discerning the Dominant Out-of-Order Performance Advantage: Is it Speculation or Dynamism*. Proceedings of the 18th international conference on Architectural support for programming languages and operating systems, ACM. 2013; 241-252.
[9] Evers M, Chang PY, Patt YN. Using Hybrid Branch Predictors to Improve Branch Prediction Accuracy in the Presence of Context Switches. In ACM SIGARCH *Computer Architecture News*. 1996; (24)2: 3-11.
[10] S McFarling. Combining Branch Predictors. *Digital Western Research Lab Technical Note* TN-36. 1993.
[11] Carole-Jean Wu, Aamer Jaleel, Margaret Martonosi, Simon C Steely Jr Joel Emer. *PACMan: Prefetch-Aware Cache Management for High Performance Caching*. MICRO. Porto Alegre, Brazil, ACM, 2011; 442-453
[12] Carole-Jean Wu, Aamer Jaleel, Margaret Martonosi, Simon C Steely Jr Joel Emer. *SHiP: Signature-based Hit Predictor for High Performance Caching*, MICRO. Porto Alegre, Brazil, ACM. 2011;430-441
[13] M Chaudhuri. *Pseudo-LIFO: The foundation of a new family of replacement policies for LLCs*. In Proceedings of the 42nd International Symposium on Microarchitecture. 2009; 401-412
[14] Mark McDermott. Queued-Stack Dataflow Processing Element for a Cognitive Sensor Platform. *International Journal of Reconfigurable and Embedded Systems (IJRES).* 2012; (1) 3: 75-86.
[15] Yang W, Guofeng Q. A Multicore Load Balancing Model Based on Java NIO. *TELKOMNIKA Indonesian Journal of Electrical Engineering*. 2012; (10) 6: 1490-1495.
[16] Intel Core i7 Processors.http://www.intel.com/products/processor/corei7/.
[17] Norman P Jouppi. *Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers.* In Proceedings of the 17th Annual International Symposium on Computer Architecture, IEEE. 1990; 364-373.
[18] Alan Jay Smith. Sequential program prefetching in memory hierarchies. *Computer*. 1978; 11(12): 7-21.
[19] I-Cheng K Chen, Chih-Chieh Lee, Trevor N Mudg. *Instruction prefetching using branch prediction information*. In Proceedings of the International Conference on Computer Design. 1997; 593-601.
[20] Glenn Reinman, Brad Calder, Todd Austin. *Fetch directed instruction prefetching*. In Proceedings of the 32nd Annual International Symposium on Microarchitecture. IEEE. 1999; 16-27.
[21] Stavros Harizopoulos, Anastassia Ailamaki. *STEPS towards cache-resident transaction processing.* In Proceedings of the 30th International Conference on Very Large Databases. 2004; (30): 660-671.
[22] Lawrence Spracklen, Yuan Chou, Santosh G. Abraham.*Effective instruction prefetching in chip multiprocessors for modern commercial applications.* In Proceedings of the 11th International Symposium on High-Performance Computer Architecture, IEEE. 2005; 225-236.
[23] Eric Sprangle, Doug Carmean*. Increasing Processor Performance by Implementing Deeper Pipelines.*Proceedings of the 29th Annual International Symposium on Computer Architecture (ISCA.02), IEEE. 2002; 25-34.
[24] Rami Sheikh, James Tuck, Eric Rotenberg. *Control-Flow Decoupling*. IEEE/ACM 45th Annual International Symposium on Microarchitecture, IEEE. 2012; 329-340.
[25] Michael Ferdman, CansuKaynak, BabakFalsafi. *Proactive Instruction Fetch*. MICRO’11, Porto Alegre, Brazil, ACM. 2011; 152-162
[26] Veynu Narasiman, Michael Shebanow, Chang Joo Lee. *Improving GPU Performance via Large Warps and Two-Level Warp Scheduling.* MICRO ’11, Porto Alegre, Brazil, ACM 978-1-4503-1053-6/11/12. 2011; 308-317.
[27] Juan L Aragón, José González, Antonio González James E Smith. *Dual Path Instruction Processing*In Proceedings of the 16th international conference on Supercomputing. ACM 1-58113-483-5/02/0006. 2002; 220-229.