Design of Instruction Analyzer with Semantic-Based Loop Unrolling Mechanism in the Hyperscalar Architecture

Yi-Xuan Lu, Jih-Ching Chiu(✉), Shu-Jung Chao, and Yong-Bin Ye

Department of Electric Engineering, National Sun Yat-Sen University, Kaohsiung, Taiwan
b0130110l2@gmail.com, chiujihc@mail.ee.nsysu.edu.tw, windy55367@gmail.com, zsefbvcx7532l@gmail.com,

Abstract. Nowadays ILP processors can't analyze the semantic information of instruction thread to change instruction series automatically for increasing ILP degree. High performance required programs such as image processing or machine learning contain a lot of loop structure. Loop structure will be bounded with the instruction number of one basic block. That cause processors are hard to enhance the computing efficiency. The characteristics of the loop structure in the program are as follows: (1) Instruction will be fetched from cache and be decoded repeatedly. (2) The issued instructions are bounded by the loop body. (3) There is data dependence between iterations. These factors will get worse the poor ILP in the loop codes. In this paper, we propose an architecture called semantic-based dynamic loop unrolling mechanism. The proposed architecture can buffer the instruction series of nested loop, unroll it automatically by analyzing the instruction flow to find the loop body with the semantic of loop instructions, store them to the instruction buffer, and dispatch them to target the processor cores. The proposed architecture consists of three units: loop detect unit (LDU), unrolling control unit (UCU) and loop unrolling unit (LUU). LDU will parse the semantic of instructions to find the closed interval of the loop body instructions. UCU will control LUU in the whole process. LUU will unroll the loop based on the information collected by LDU. Loop controller will handle the complementation overhead for branch miss prediction and the loop finish-up codes. The verifications use ARM instructions generated by *Keil µVision5* compiler. The results show that eliminating iteration dependence can improve ILP by 140% to 180%.

Keywords: ILP of loop · Semantic of loop · Loop unrolling · Hyperscalar · Nested loop

1 Introduction

Loop structures are the main portion of program [1]. The characteristics of the loop structure are as follows: (1) Instruction will be fetched from cache and be decoded again and again. (2) The repeat dependence of instructions in the loop body. (3) The dependence relations between iterations. These factors will cause poor ILP in the

C.-Y. Chang et al. (Eds.): ICS 2018, CCIS 1013, pp. 3–19, 2019.
https://doi.org/10.1007/978-981-13-9190-3_1

implementation of the loop for the super-scalar architecture. To improve the computing efficiency of super-scalar architecture, and combine the characteristic of it. In this paper, we propose an approach, called semantic analyzer for loop unrolling, which can increase ILP of loops by parsing the semantics of instructions for collecting the required information of loop unrolling. Loop structure has a specific ordering pattern in machine codes, which produced by compiling it, by formulating the semantic of the loop with the observations of this pattern, we can find the section of loop.

In this paper, we build a semantic-based dynamic loop unrolling mechanism on the instruction analyzer in hyper-scalar architecture, we exploit the ILP for loop structures by unrolling and eliminating iteration of the loop. The characteristics of the semantic-based dynamic loop unrolling mechanism are as follows: (1) Parsing the semantic of instructions to find the closed interval of the loop body instructions. (2) Promote the ILP of loop instructions by eliminating its iteration dependence with an immediate operation. (3) Analyzing the situation during loop unrolling and the relationship between loops to achieve unrolling of a nested loop. (4) Update the data dependence tag of instructions when the branch instruction is taken. (5) Flush the instructions which should not be executed when the branch instruction taken happened. The concepts of proposed architecture are shown in Fig. 1.

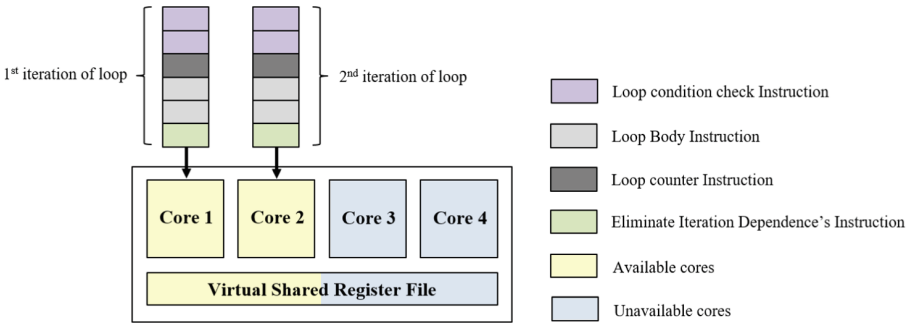


Fig. 1. Concepts of semantic-based dynamic loop unrolling mechanism

2 Related Work

In Hyper-scalar architecture [2–6], it allows the multi-core system to allocate the cores in the system into a single processor system to accelerate of one program. The characteristics of Hyper-scalar architecture are as follows: 1. It can group cores in processor dynamically. 2. its architecture is high flexibility and scalability, 3. It can accelerate single-threaded performance with several cores.

The instructions in Hyper-scalar architecture can be divided into two types as follows by the dependence between them: 1. Intra-Dependence. 2. Inter-Dependence. Hyper-scalar architecture solves Inter-Dependence by analyzing the dependence between instructions dynamically and establishing a distributed system to exchange information between cores.

Figure 2 shows the architecture of Hyper-scalar. The Hyper-scalar architecture dispatch instructions into cores based on the current hardware resources. To exchange information from cores, Hyper-scalar architecture finds the data dependence of instructions clearly by analyzing the relationship between instructions. In order to communicate the information required by the instructions to each core correctly, Hyper-scalar architecture proposed a distributed exchanging information system. It adds information processing unit in each core, the unit is built to deal with the data request from other cores and dispatch the request of the core itself to each core. By connecting cores in the processor, an information exchanging network is built.

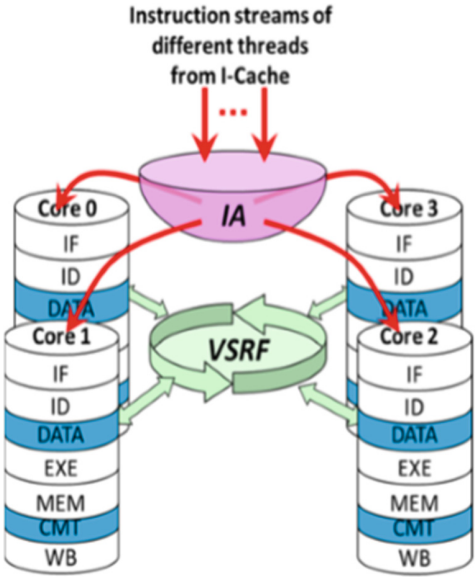


Fig. 2. Hyper-scalar architecture

Hyper-scalar architecture analyzes the dependence of instructions by instruction analyzer (IA). IA analyzes the relationship between instructions and generates the dependence tag of instructions. Virtual shared register files (VSRF) deals with the information exchanging between cores by dependence tag.

Comparing Hyper-scalar architecture and super-scalar architecture, both of them can analyze the dependence of instructions, and dispatch the instruction which data are prepared. By building an information exchanging system of cores, Hyper-scalar architecture has a better performance than super-scalar architecture, but it still has a poor performance when facing loop structure [7–10].

3 Dynamic Loop Unrolling Mechanism

This paper proposed the dynamic loop unrolling mechanism based on semantic analysis of nested loop can analyze the semantic information of instructions and find the interval of a loop. It can also unroll loop and dispatch to each core to improve the ILP as Fig. 3 shows.

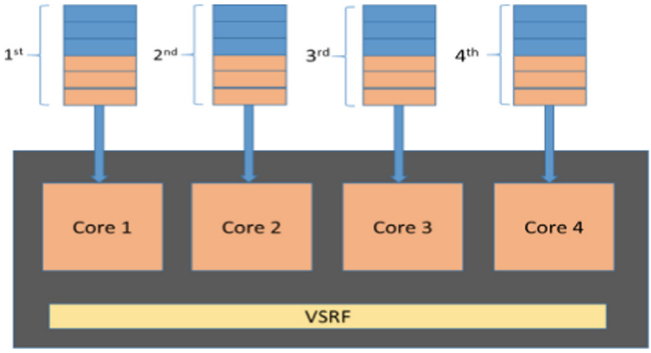


Fig. 3. Loop unrolling

To represent the instruction flow of loop structure, we define six types of nodes: Jump node, Normal node, Counting times node, Flag set node, Branch node, Initial time node, shown as Fig. 4.

Fig. 4. Six types of node

Observe instruction flow of loop, will find the law of compiler, shows as Fig. 5. With the law, we can collect the information in each layer of loop.

3.1 System Architecture

The system architecture proposed by this paper shows as Fig. 6. It can be divided into three parts:

3.1.1 Loop Detect Unit (LDU)

Loop detect unit located in the IA, it analyzes the information came from Pre-Decoder to find the instruction interval of loop. When finding the interval of loop, it will record the Loop Address Information into Loop Buffer inside itself. Loop detect unit compares the instruction which was caught from Instruction Cache with detected loop interval, if

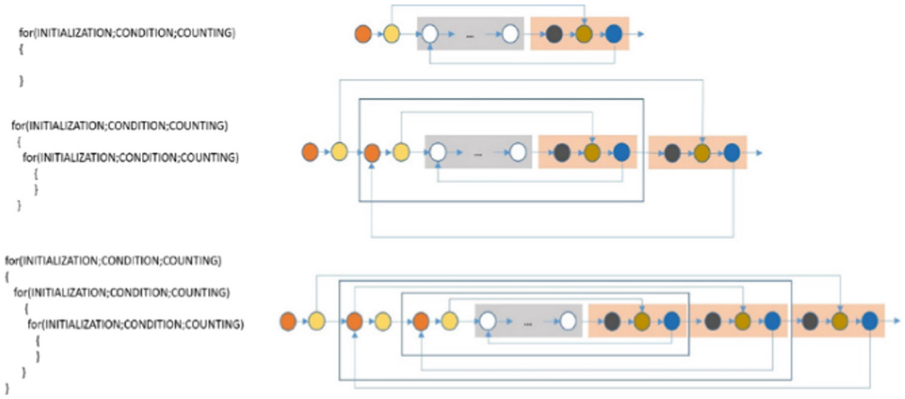


Fig. 5. The instructions of loop after compiled

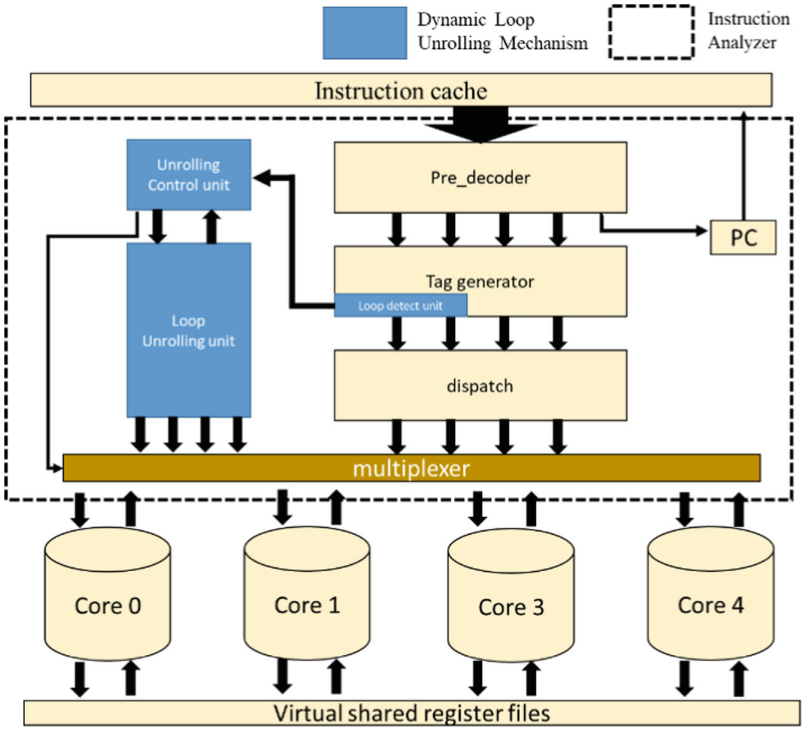


Fig. 6. System architecture

the address is in the interval, it will record the instruction and its operand frame from Pre-Decoder into Loop Instruction Table and Parsing Table.

When Loop Address Information was collected completely, loop-detect-unit will compare the information with other loop's address information to find nested loop

structure in the instruction flow. Loop detect unit also record the relationship between loops. After all information in the loop interval in the instruction flow is collected completely, it will send a signal to unrolling control unit (UCU) and dispatch the instruction and its operand frame to UCU according to its request.

3.1.2 Unrolling Control Unit (UCU)

Unrolling control unit is designed to deal with unrolling nested loop. It sends a data request to loop detect unit to get instructions and its operand frame from Loop Instruction Table and Parsing Table and dispatches to loop unrolling unit (LUU) before start unrolling. The dynamic loop unrolling mechanism proposed by this paper divided nested loop structure into inner layer loop and outer layer loop. Dynamic loop unrolling mechanism unrolls the outer layer loop first to get the number of executions of outer layer loop, then unrolls the inner layer loop continuously until the completed times of unrolling equals to the number of executions of outer layer loop. Unrolling control unit decides the number of unrolling loop according to the relationship between loops and the unrolling condition.

3.1.3 Loop Unrolling Unit (LUU)

Loop unrolling unit is designed to deal with loop unrolling, it promotes ILP by eliminating the iteration dependence. Loop unrolling unit detects and eliminates the iteration dependence with eliminating iteration dependence unit, when getting the parsed operand frame from UCU. After eliminating the iteration dependence, loop unrolling unit will generate dependence tag and loop tag to record the execution times of loop. The generated tag and instruction will be pushed into the instruction dispatch queue, and wait to be dispatched to cores.

If unrolling is completed, dynamic loop unrolling mechanism needs to update the data dependence of VSRF mapping table and memory tag mapping table in IA. It updates the data dependence with loop VSRF mapping table and loop memory tag mapping table in the loop unrolling unit.

3.2 Loop Detect Unit

Loop detect unit finds the loop interval in instruction flow by analyzing the semantic information of instruction flow, and dispatches instructions and parsed operand frames according to the request from UCU, the architecture shows as Fig. 7.

The semantic of loop structure in the instruction flow is detected as follow:

Step 1: When detect absolutely jump instruction in the instruction flow, then record the jump address (JA) as loop start address (LS) and its jump target address (JTA) as loop body end address (LBE), shows as Fig. 8. Get into the next step.

Step 2: When detect branch instruction in the instruction flow, then record the branch address (BA) as loop end address (LE) and branch target address (BTA) as loop body start address (LBS), shows as Fig. 9. Get into the next step.

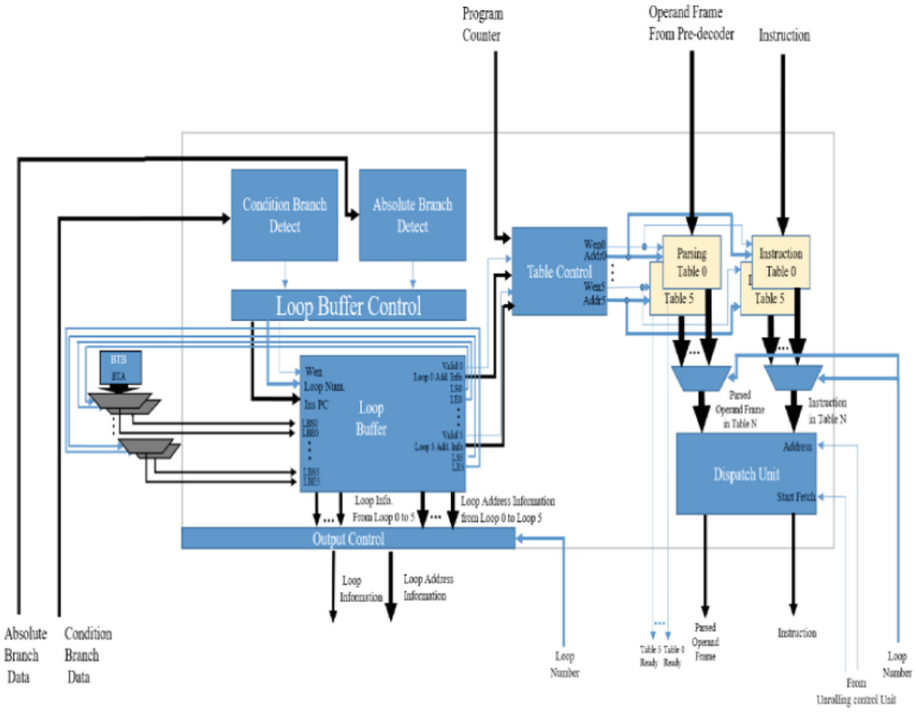


Fig. 7. Architecture of LDU

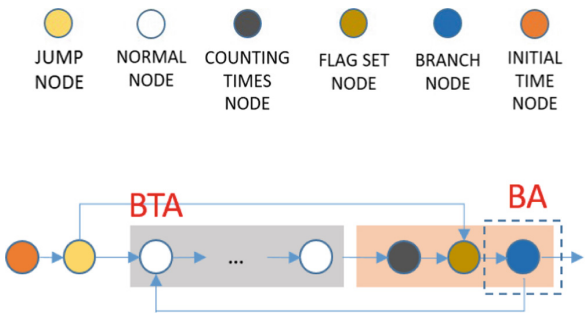


Fig. 8. Loop detecting-Step 1

Step 3: If the information satisfies following conditions: JA < BTA < JTA < BA and BTA = JA + 1. The interval between JTA and BTA is defined as a loop body, shown as Fig. 10.

Each Loop Buffer contains two tables to store the instruction and its parsed operand frame in the interval of the loop. (1). Loop Instruction Table:Record the machine code of the instructions in the interval of loop. (2). Loop Parsing Table:Record the parsed operand frame of the instructions in the interval of loop, shows as Fig. 11.

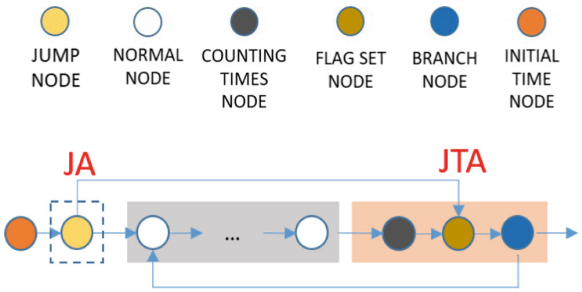


Fig. 9. Loop detecting-Step 2

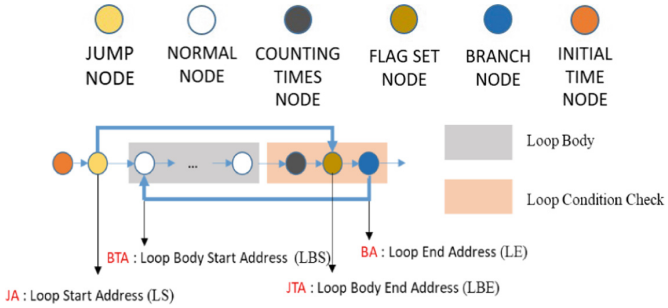


Fig. 10. The interval of loop

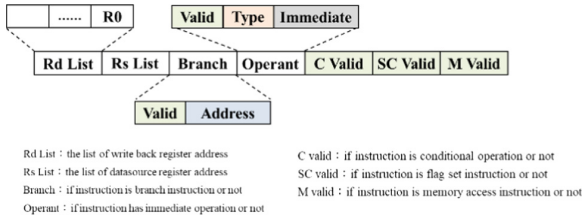


Fig. 11. The format in loop parsing table

3.3 Unrolling Control Unit

Unrolling control unit decides the number of loop by analyzing the signals from LDU and the relationship between loops in unrolling table, the architecture shows as Fig. 12.

The dynamic loop unrolling mechanism proposed by this paper divides the nested loop into outer layer and inner layer when unrolling it as Fig. 13 shows. Dynamic loop unrolling mechanism unrolls the outer layer loop first to get the number of executions of outer layer loop, then unrolls the inner layer loop continuously until the completed times of unrolling equals to the number of executions of outer layer loop. Unrolling the outer layer loop is for getting the number of executions. Therefore, UCU has to deal with the information which will be dispatched to LUU.

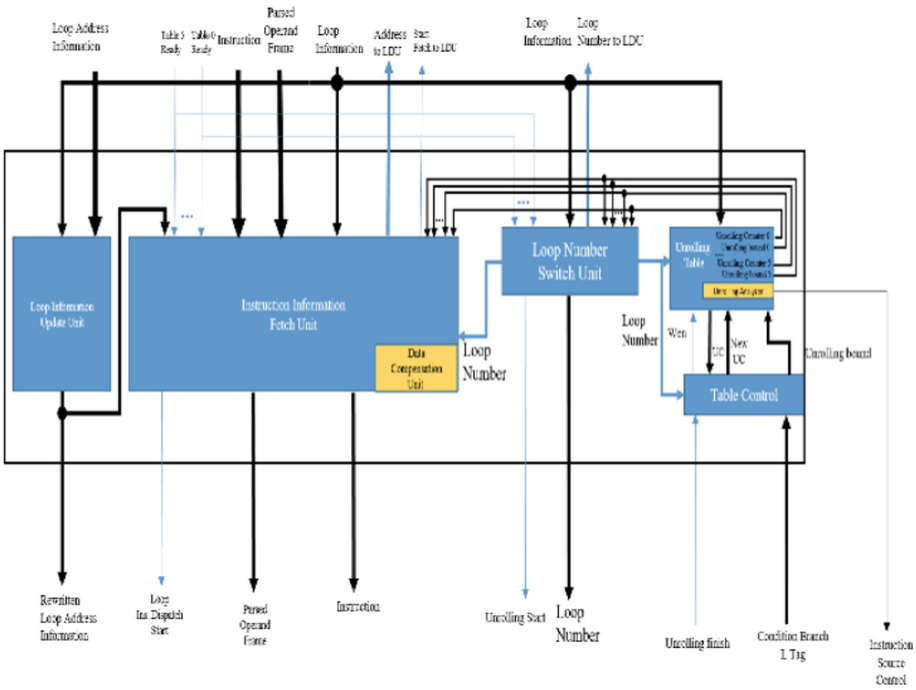


Fig. 12. Architecture of UCU

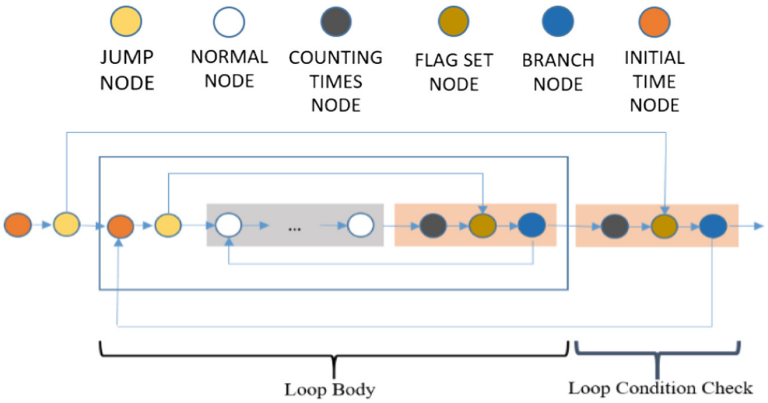
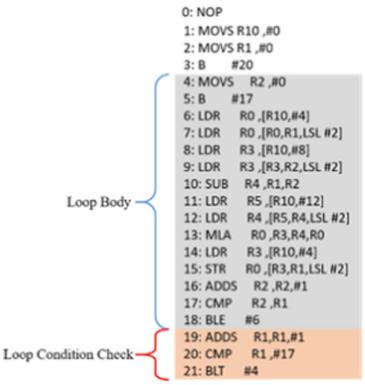


Fig. 13. The structure of nested loop

Take a two-layer nested loop, for example, its loop body contains a single loop. Unrolling the interval of loop condition check to get the number of executions of outer layer loop. To get the correct interval and loop information, when finding the loop in the loop buffer is nested loop, change the loop body start address of outer layer loop into the start address of its loop condition check as Fig. 14 shows.



Loop Buffer							
Number	valid	LS	LBS	LBE	LE	LIN	MLF
0	1	3	19	20	21	1	1
1	1	5	6	17	18	-	-
2	-	-	-	-	-	-	-

Loop Buffer							
Number	valid	LS	LBS	LBE	LE	LIN	MLF
0	1	3	4	20	21	1	1
1	1	5	6	17	18	-	-
2	-	-	-	-	-	-	-

Fig. 14. The correction of nested loop

Unrolling table is inside UCU, it records the unrolling process. Unrolling table records the number of executions of outer layer loop as unrolling bound and records the times of unrolling complete of inner layer loop as unrolling counter.

When the outer layer loop unrolls completely, UCU records the execution times in the unrolling table by its LIN in the loop buffer. When inner layer loop unrolls completely, UCU records the times in the unrolling counter of unrolling table by its own loop number, as Fig. 15 shows.

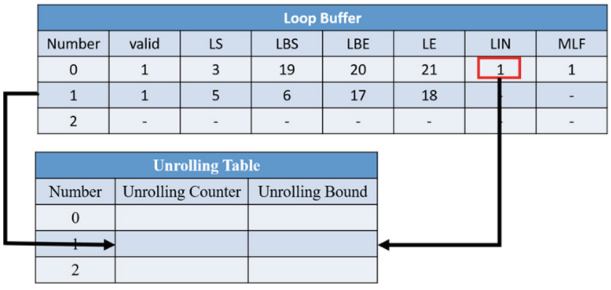


Fig. 15. Data updating in unrolling table

3.4 Loop Unrolling Unit

Loop unrolling unit (LUU) decides the times of loop unrolling by available core numbers. LUU generates the dependence tag of instructions and the dependence eliminated iteration with immediate operation by the information collected after unrolling finish. It also generates mapping tables to compensate for the dependence of some instructions been aborted. Rearrange dispatch order of these instructions before dispatch, the architecture shows as Fig. 16.

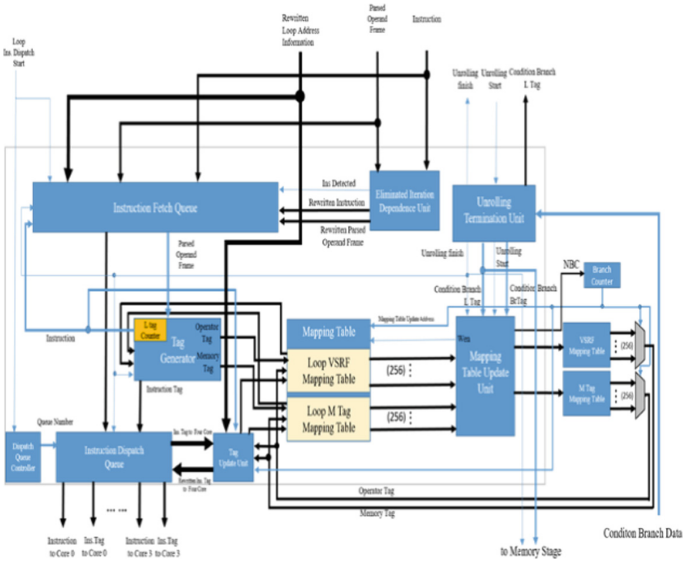


Fig. 16. Architecture of LUU

LUU deals with the data dependence between iterations by register renaming. The RAW hazard of instructions in the internal loop might cause to get the incorrect data dependence in this renaming method, example shows as Fig. 17.

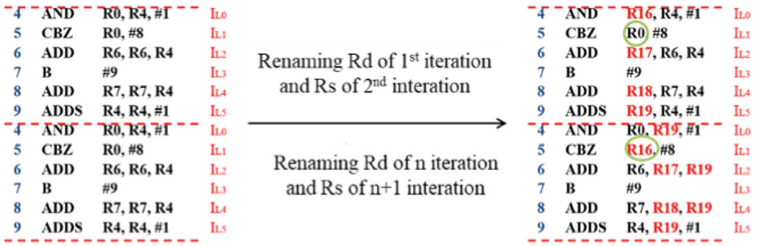


Fig. 17. Register renaming between iterations

To deal with it, LUU find out those instructions with the RAW hazard of internal loop and rename them later, shows as Fig. 18.

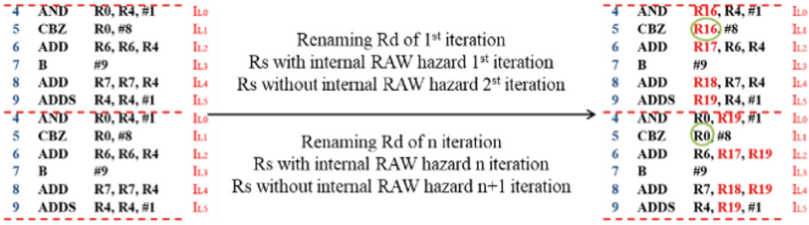


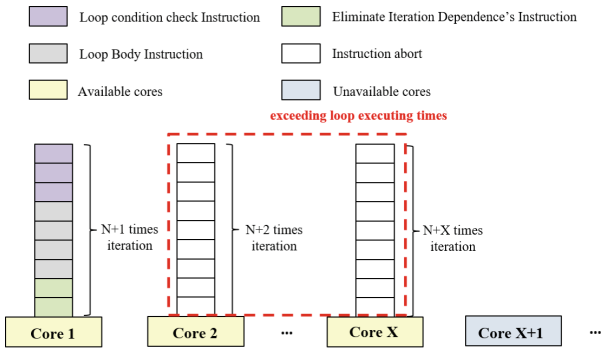
Fig. 18. Register renaming between iterations

There are some situations may cause wrong data dependence mapping or memory get the wrong data, hen unrolling loop. This paper proposed an architecture with compensation mechanism to make sure the accuracy of data dependence mapping and data writing into memory. The compensation mechanism has been divided into two parts to deal with two situations which shown in Fig. 19.

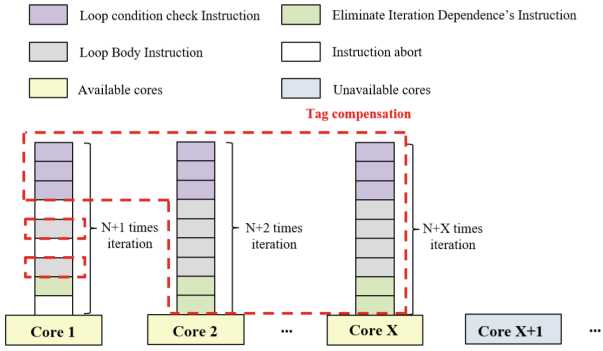
1. The exceeding of loop execution times
2. Specific instruction flush

Since some instructions will be aborted, the data dependence tag of the instructions after the instructions which be aborted might be wrong. To deal with it, we record the data dependence mapping situation demarcated by the branch instruction. Those data dependences are recorded in compensation tables such as Loop VSRF Mapping Table, Loop M Tag Mapping Table and Specific Instruction Flush Table by adding the basic value in Up-to-date Mapping Table to the offset tag of those tables which generated by LUU, shows as Fig. 20.

The L tag records the iteration times of instructions. When the exceeding of loop execution occurs, the mapping tables of IA will be updated by L tag before returning dispatch right to IA.



(A) The exceeding of loop execution times



(B) Specific instruction flush

Fig. 19. Compensation mechanism

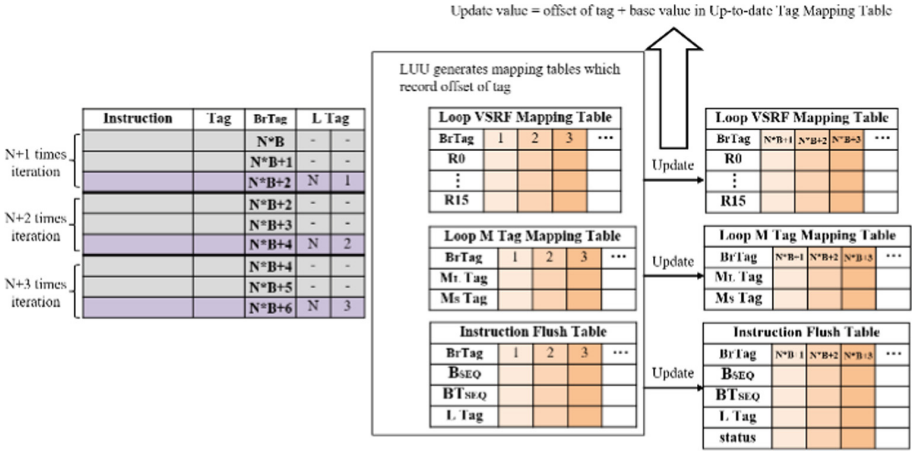


Fig. 20. Compensation tables

Memory Access Delay Buffer is designed to make sure memory data accuracy, shows as Fig. 21. When memory load occurs, it will be searched first, and then get data from memory if there is no data in it. The memory data will store data to Memory Access Delay Buffer first and then store back to memory according to L tag when the unrolling is done.

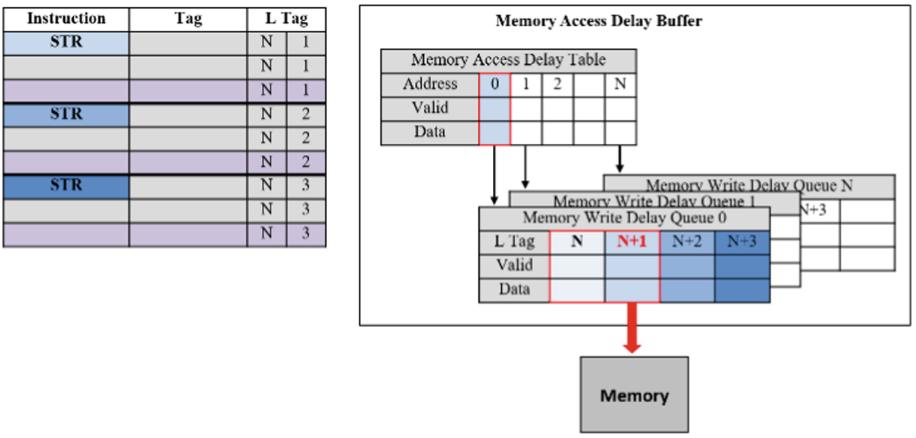
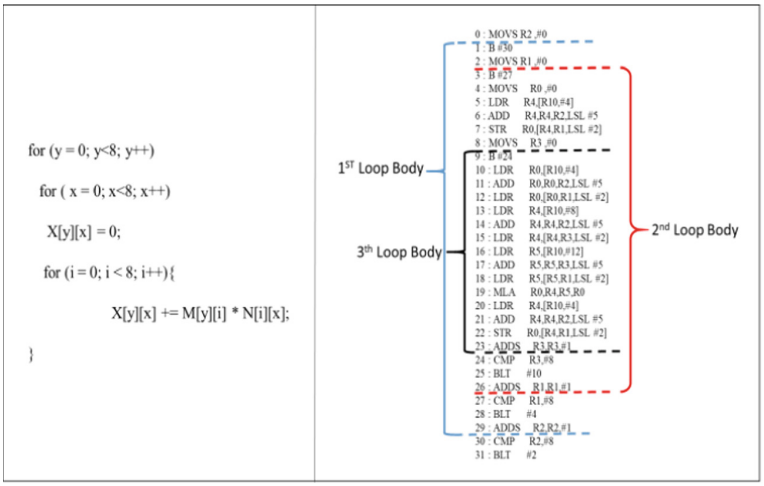


Fig. 21. Memory access delay buffer

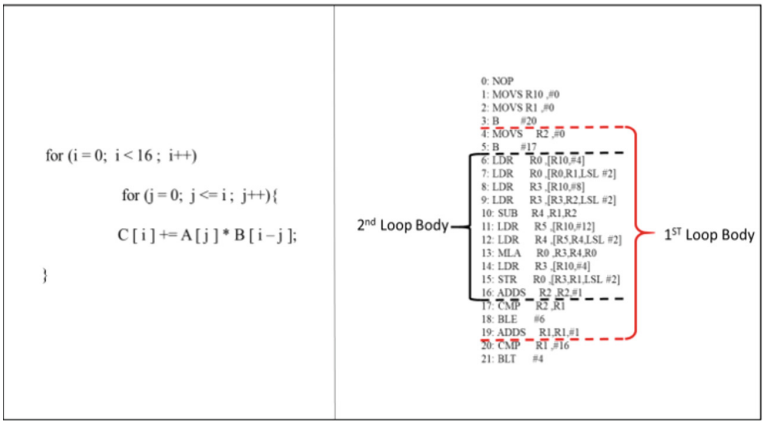
4 Simulation and Results

Simulation use C language to build a software simulation model to verify the implementation of semantic-based dynamic loop unrolling mechanism in the Hyper-scalar architecture. The parameter of the simulation model is as shown in Table 1. And the test programs generated by Keil-μVision-5 compiler are as shown in Fig. 22.

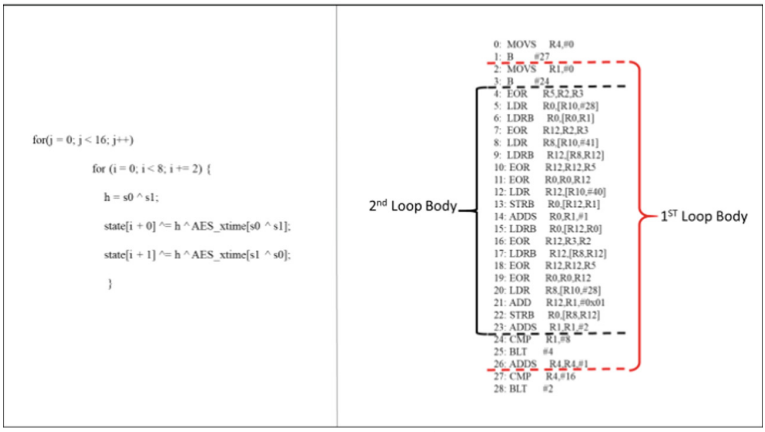
The results of instruction flow simulation shown in Fig. 23. The results show that eliminating iteration dependence can improve ILP by 140% to 180%, because of the redundant instructions generated during unrolling, performance improved 50% to 100%.



(A) Matrix multiplication



(B) Convolution



(C) AES-mix column

Fig. 22. The test program

	With unrolling mechanism	Without unrolling mechanism
instructions	5027	3140
cycles	1257	2062
ILP	1.570381	0.591049

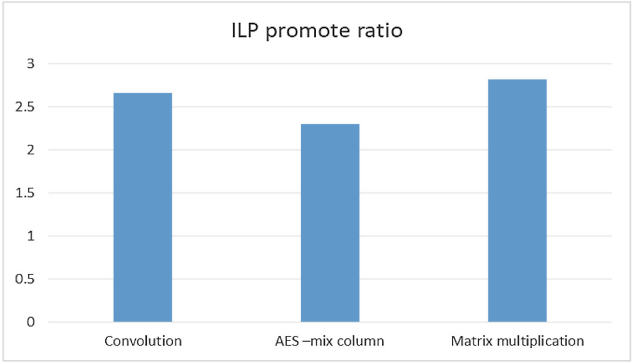
(A) Convolution

	With unrolling mechanism	Without unrolling mechanism
instructions	4621	3108
cycles	1123	1640
ILP	1.607916	0.698312

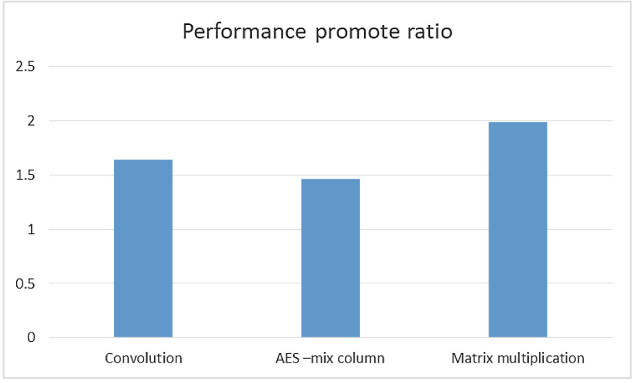
(B) AES-mix column

	With unrolling mechanism	Without unrolling mechanism
instructions	21757	14408
cycles	5703	11323
ILP	1.748356	0.621463

(C) Matrix multiplication



(D) ILP promote ratio



(E) Performance promote ratio

Fig. 23. Simulation results

5 Conclusion and Future Work

This paper proposes the semantic-based dynamic loop unrolling mechanism which is based on the hyper-scalar architecture, it can detect and unroll the loop automatically. It can also decide the unrolling time by core resource.

By eliminating iteration dependence and specific instruction flush can promote ILP of loop. It also makes sure the accuracy of data dependence compensation mechanism. By the result of software simulation, we know that the semantic-based dynamic loop unrolling mechanism can increase the performance while executing loop. When facing the unfixed repeated times of loop and the iteration with the branch instruction, this architecture can also make sure the accuracy of data. In this mechanism, we unroll the loops only according to how many cores resources there are. It may cause the redundant instructions generated during unrolling. We must release the core resources dynamically [11], so that its unrolling time will depend on how many times it used to execute to increase the efficiency of the processors.

References

1. Rotenberg, S.B., Smith, J.E.: Trace cache: a low latency approach to high bandwidth instruction fetching. In: Proceedings of the 29th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO-29, pp. 24–34 (1996)
2. Chou, Y.-L.: Study of the hyperscalar multi-core architecture, Department of Electrical Engineering National Sun Yat-Sen University (2011)
3. Su, D.-S.: Design of the execution-driven simulation environment for hyper-scalar architecture, Department of Electrical Engineering National Sun Yat-Sen University (2008)
4. Chiu, J.-C., Chou, Y.-L., Chen, P.-K., Ding-Siang, S.: A unitable computing architecture for chip multiprocessors. Comput. J. **54**(12), 2033–2052 (2011)
5. Chen, P.-K.: ESL model of the hyper-scalar processor on a chip, Department of Electrical Engineering National Sun Yat-Sen University (2007)
6. Chiu, J.-C., Huang, Y.-J., Ye, Y.-L.: Design of the optimized group management unit by detecting thread parallelism on the hyperscalar architecture, National Computer Symposium, December 2013
7. Yeh, T.Y., Marr, D.T., Patt, Y.N.: Increasing the instruction fetch rate via multiple branch prediction and a branch address cache. In: 7th International Conference on Supercomputing, pp. 67–76, July 1993
8. Dennis, J.B., Misunas, D.P.: A preliminary architecture for a basic data-flow processor. In: Proceedings of the 2nd Annual Symposium on Computer Architecture, Houston, TX, pp. 126–131, January 1975
9. Lerner, E.J.: Data-flow architecture. IEEE Spectr., 57–62 (1984)
10. Fisher, J.A., Faraboschi, P., Young, C.: Embedded Computing, A VLIW Approach to Architecture, Compilers and Tools. Elsevier (2005)
11. Huang, Y.-J.: Design of the optimized group management unit by detecting thread parallelism on the hyperscalar architecture, Department of Electrical Engineering National Sun Yat-Sen University (2013)