Computer Engineering and Intelligent Systems
ISSN 2222-1719 (Paper) ISSN 2222-2863 (Online)
Vol 3, No.4, 2012

Dynamic Instruction Scheduling For Microprocessors Having Out Of Order Execution

Suresh Kumar, Vishal Gupta*, Vivek Kumar Tamta

Department of Computer Science, G. B. Pant Engineering College, Pauri, Uttarakhand, INDIA

* E-mail of the corresponding author: vishalgupta87@gmail.com

Abstract
Dynamic Instruction Scheduling is very much needed for fast working of multiprocessor and reduction of overhead by the processor. The Instruction scheduling logic mainly depends on associative searching of the entries to the dynamic wakeup instructions for the execution. We also describes the scheduler concept which also the concern for scalability and complexity of the multiprocessor. We have different Dynamic Instruction Scheduling Logic highlighting the objectives, goals, advantages and challenges facing during scheduling logic like energy issues and complexity issues as well as full description of dynamic instruction Scheduling logic. In this paper, we will be presented in a comprehensive analysis to reschedule the execution order of instructions for improve the performance of microprocessor.
Keywords: Dynamic Instruction Scheduling, Instruction Grouping, Issue Queue.

1. Introduction

General purpose microprocessors usually apply superscalar and out-of-order execution. These microprocessors should dynamically extract instruction level parallelism (ILP) for high performance because they are required to execute various types of programs efficiently and must also have to run a number of legacy binary codes. However, dynamic instruction scheduling logic for out-of-order execution has a serious problem in that it consumes a significant amount of energy due to the complicated nature of its hardware logic. We propose a micro architectural technique and hardware implementation to reduce the complexity and energy consumption of dynamic instruction scheduling logic by grouping instructions together in the instruction queue. The concept of the proposed method is to group several instructions together and let the dynamic instruction scheduling logic treats them as a single instruction. Thus, grouping should be performed in the dispatch stage when instructions are written into the instruction queue. In the present paper, we propose the grouping of two instructions by using dependence information. If issuing one instruction is the only requirement for starting the other instruction, this pair will be grouped together. By treating the grouped instructions as a single instruction, dynamic instruction scheduling logic becomes capable of holding and issuing a greater number of instructions without increasing the size or number of ports of the instruction queue or the selection logic.

2. Logic behind Scheduling

This shows the conventional dynamic instruction scheduling logic of a four-way out-of-order superscalar processor. The right hand side of the figure shows the instruction queue, and the blue line indicates a single entry, which is composed of the CAM logic for tag matching and the payload RAM (op codes, physical register number, offsets, etc.). The left-hand side shows the selection logic.

The instruction queue has four read and write ports for dispatch and issue, respectively, and the selection logic has the ability to select four instructions per cycle.

3. Proposed Technique

The objective of this study is to reduce the complexity and energy consumption of dynamic instruction

scheduling. We propose the grouping of several instructions together in order to reduce the required hardware of dynamic instruction scheduling.

A. Grouping Condition

By grouping two instructions, waking up, selecting, and issuing only the first instruction leads to the issuing of both instructions because the second instruction should be issued in the cycle following the first instruction. We decided to group a single latency operation for the first instruction because this can simplify the logic and the hardware used to support the proposed technique, which is described in Section C. The candidate of grouping, the latency of which is single cycle, is an integer ALU operation. We are required to implement the proposed technique in the instruction queue, in which the integer instructions are dispatched. The details of hardware implementation of the proposed dynamic instruction scheduling logic are described in Section C. Instruction pairs, which are candidates of the proposed grouping technique, can be classified into the following two types.

[I] The issuing of one instruction is the only requirement for issuing the other instruction.

[II] Both instructions can be issued in the same cycle.

We first discuss about Type [I], which consists of two different types of pairs, each of which will be explained using an example.

Instruction A: add r5 ← r3, r2

Instruction B: add r4 ← r5, R (1)

We can see that right operand of Instruction B is ready, and the left operand r5 is the destination of Instruction A. Therefore, only the condition of issuing Instruction A enables Instruction B to be issued in the next cycle.

The pair of instructions below also belongs to Type [I].

Instruction A : add r5 ← r3, r2

Instruction B : add r4 ← r5, r3 (2)

The left operand of Instruction B is the destination of Instruction A and is the same as Example (1). Furthermore, the right operand r3 of Instruction B is the left operand of Instruction A, so if Instruction A is ready for issue, r3 is also ready. For this reason, the above instructions can be grouped together, as in the case of Example (1). Next, we describe Type [II]. The instructions of Example (3), given below, are both ready for issue, because both the left and right operands are ready. Since both instructions are ready, they can be grouped, although there are no data dependencies between these two instructions. In this case, the previous instruction in program order is issued first. When grouping is not performed, these two instructions might be able to be executed in the same cycle. Therefore, the second instruction will be issued one cycle later and may cause performance degradation. However, in the proposed technique, it is important to obtain a high throughput by grouping as many instructions as possible. Moreover, ready instructions usually stay in the instruction queue for only a short time, and most of these instructions are not critical to the performance. Thus, even if the second instruction is issued one cycle later, there is almost no performance degradation. Therefore, we group ready instructions. Although more patterns that belong to the above two types exist, we decided to group only three types of pairs described above.

Instruction A : add r1 ← R, R

Instruction B : add r2 ← R, R (3)

B. Improved Implementation

We resolve the low flexibility of the dispatch stage, which is a fundamental problem of the dynamic instruction scheduling of Figure 1.

4. Experimental Setup

We used the Simple Scalar Tool Set as the base simulation environment. We extended Simple Scalar so that the proposed microarchitecture was evaluated. For estimating the energy consumption, we used the Wattch extension. We used all of the programs of the SPEC CPU2000 integer benchmark suite. The programs were compiled by the DEC C compiler for Alpha AXP instruction set architecture (ISA). We fast-forwarded two billion instructions and simulated 200 million instructions for all of the evaluated programs.

Table 1 shows the processor configuration assumptions used for the evaluation. In order to group several instructions, we assumed an integer-load/store queue. Both the integer and load/store instructions are dispatched to this queue. By this assumption, we can group not only two integer ALU operations, but also the integer ALU operation for the first instruction and a load instruction for the second instruction. The size of the integer-load/store queue was varied throughout the evaluation from 16 to 128. We ignored the energy consumption of the additional hardware as negligible.

5. Result

A. Performance

As seen from the figure 3, the IPC degrades gradually as the instruction queue size decreases. This is because the number of instructions to execute in parallel in order to exploit ILP is limited by the lack of free entry of the instruction queue.

B. Energy Consumption

The proposed technique saves energy due to the reduced instructions to select and issue, which simplifies the instruction scheduling logic. Figure 4 presents the average energy savings of the instruction scheduling logic of FULL (denotes new proposed logic). In 48-entry FULL, the energy saving was 56%, 7%, 52%, and 58% for dispatch, wake-up, select, and issue, respectively, compared to the conventional dynamic instruction scheduling logic. Except for the wake-up stage, the reduction in energy consumption is large. In the case that energy consumption of the wake-up stage is dominant; we should consider halving the number of entries of the proposed implementation in order to reduce the energy consumption of the wake-up logic.

7. Conclusion

In the present work, we introduced a dynamic instruction scheduling concept that groups several instructions. The proposed technique enables that logic to hold and issue more instructions without increasing the size or number of ports and energy consumption is also reduced by grouping the instructions.

References

T. M. Austin, E. Larson, and D. Ernst. Simple scalar: An infrastructure for computer system modeling. IEEE Computer, 35(2):59–67, 2002.

Hiroshi Sasaki, Masaaki Kondo and Hiroshi Nakamura. Energy-Efficient Dynamic Instruction scheduling logic through instruction grouping. IEEE Transactions on VLSI Systems, Vol. 17, No. 6, 2009

A. Buyuktosunoglu, T. Karkhanis, D. H. Albonesi, and P. Bose. Energy efficient co-adaptive instruction fetch and issue. In ISCA, pp. 147–156, 2003.

R. Canal and A. Gonz´alez. Reducing the complexity of the issue logic. In ICS, pp. 312–320, 2001.

D. Folegnani and A. Gonz´alez. Energy-effective issue logic. In ISCA, pp. 230–239, 2001.

P. Michaud and A. Seznec. Data-flow prescheduling for large instruction windows in out-of-order

processors. In HPCA, pp. 27–36, 2001.

J. J. Sharkey and D. V. Ponomarev. Efficient instruction schedulers for smt processors. In HPCA, 2006.

J. J. Sharkey, D. V. Ponomarev, K. Ghose, and O. Ergin. Instruction packing: reducing power and delay of the dynamic scheduling logic. In ISLPED, pp. 30–35, 2005.

S. A. Taylor, M. Quinn, D. Brown, N. Dohm, S. Hildebrandt, J. Huggins, and C. Ramey. Functional verification of a multiple-issue, out-of-order, superscalar alpha processor - the dec alpha 21264 microprocessor. In DAC, pp. 638–643, 1998.

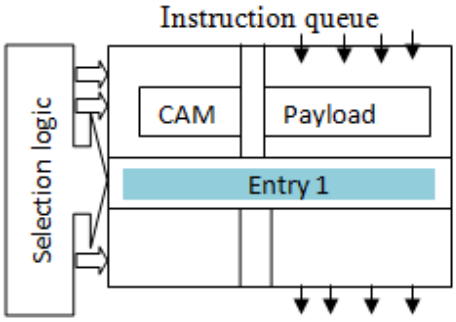

Figure 1: Conventional dynamic scheduling

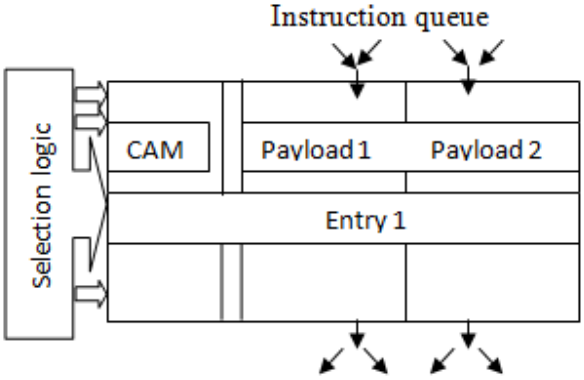

Figure 2: Showing grouping of Instructions

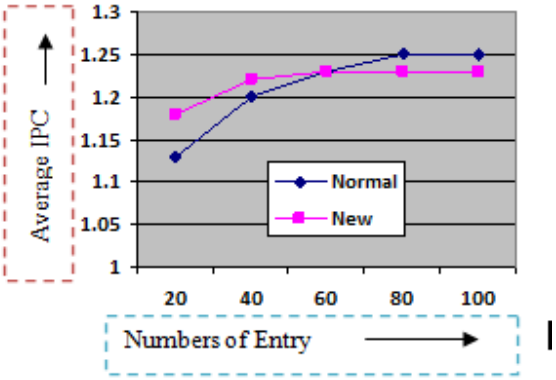

Figure 3: Average IPC

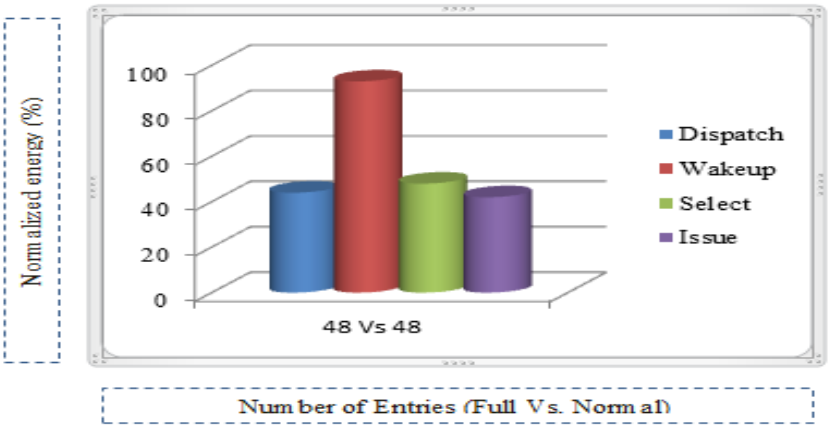

Figure 4: Average Energy of FULL

Fetch & Decode width	4
Branch prediction	Combined bimodal (4K-entry) gshare (4K-entry), selector(4K-entry)
BTB	1,024 sets, four-way
Mis-prediction penalty	three cycles
Reorder buffer size	96
Memory latency	100 cycles
Bus width	16B
Bus clock	1/4 of processor the core
Instruction queue size - floating-point	32
Commit Width	4

Table 1: Processor configuration