A Super Instruction-Flow Architecture for High Performance and Low Power Processors

Kenji Kise, Takahiro Katagiri, Hiroki Honda, and Toshitsugu Yuba

Graduate School of Information Systems,
The University of Electro-Communications

Abstract

Microprocessor performance has improved at about 55% per year for the past three decades. To maintain this performance growth rate, next generation processors with more than one billion transistors must achieve higher levels of instruction level parallelism. However, it is known that a conditional branch poses serious performance problems in modern processors. In addition, as an instruction pipeline becomes deep and the issue width becomes wide, this problem becomes worse. The goal of this study is to develop a novel processor architecture which mitigates the performance degradation caused by branch instructions. In order to solve this problem, we propose a **super instruction-flow architecture**. This architecture has a mechanism which processes multiple instruction-flows efficiently and tries to mitigate the performance degradation. Preliminary evaluation results with small benchmark programs show that the first generation super instruction-flow processor efficiently mitigates branch overhead.

1 Introduction

In recent years, in order to improve processor performance, various speculation techniques, such as data value prediction and dynamic memory disambiguation, have been proposed. Although these speculation techniques are becoming indispensable for high performance processors, a new paradigm or new architecture is required to attain the dramatic boost of available instruction level parallelism. In this context, new architectures such as instruction level distributed processing [2], the grid processor [4] and the TRIPS processor [7] are being examined. Although these architectures offer a powerful execution mechanism, a sophisticated instruction fetch mechanism which supplies sufficient instructions is inevitable. A scalable instruction fetch mechanism still remains an important research topic.

Most of the high performance processors in the market predict control-flow using a sophisticated branch predictor [8]. However, even if a processor uses one of the latest branch predictors, such as YAGS [1], hybrid branch predictors or two-level adaptive branch predictors, the prediction accuracy is about 95% at most. A branch predictor cannot avoid the misprediction of a fixed rate. In order to reduce the overhead of a branch instruction, in addition to the effort to reduce the number of mispredictions, the reduction of misprediction penalties becomes important.

The goal of this study is to develop a novel processor architecture which mitigates the performance degradation caused by branch instructions. In order to solve this problem, we propose a **super instruction-flow architecture**. This architecture is a type of decoupled architecture [9]. It has a mechanism to process multiple instruction-flows efficiently in order to mitigate performance degradation. In addition to the architectural proposal, we describe the design of its first generation processor and report our preliminary evaluation results with small benchmark programs. Our target applications are single-threaded programs. Multi-threaded programs are not in our current research scope.

2 Super Instruction-Flow

Super instruction-flow is proposed as the new architecture for attaining high instruction level parallelism and low power consumption. It is a type of decoupled architecture [9]. It has a mechanism to process multiple instruction-flows efficiently in order to mitigate the performance degradation caused by branch instructions. A comparison of the instruction pipeline of a super instruction-flow processor and a conventional superscalar processor is shown in Figure 1.

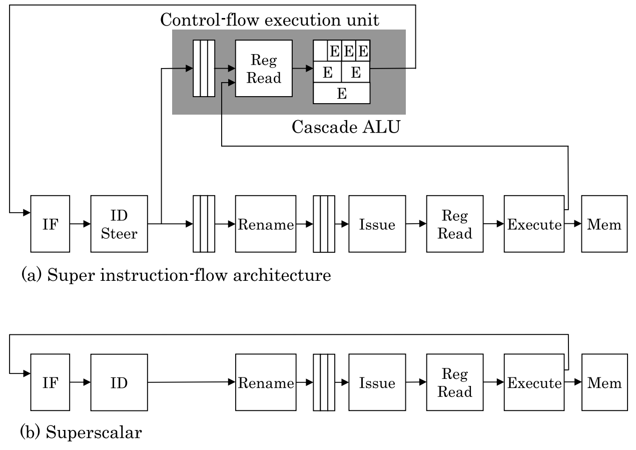

Figure 1: Comparison of the instruction pipeline of a super instruction-flow processor and superscalar processor.

Branch instruction and branch-related instruction, including the calculation of a branch condition, are called **control-flow instructions**. Other instructions are called **data-flow instructions**. In an early stage of the instruction pipeline, a control-flow instruction and a data-flow instruction are classified. Then, the control-flow instruction is handled at high speed by the dedicated hardware. The shaded portion of Figure 1 is the dedicated hardware, which is named the control-flow execution unit. By classifying the control-flow instruction and data-flow instruction, the optimizations suitable for each flow can be applied. The penalty of a branch misprediction is mitigated by handling the control-flow instruction with priority. The super instruction-flow architecture is unique to merge the Cascade ALU architecture[5] on the control flow execution unit.

2.1 First Generation Processor

The block diagram of the first generation super instruction-flow processor is shown in Figure 2. In the instruction fetch stage, some instructions are fetched from the instruction cache every cycle. The number of instructions to be fetched per cycle and the instruction fetch width are important processor parameters.

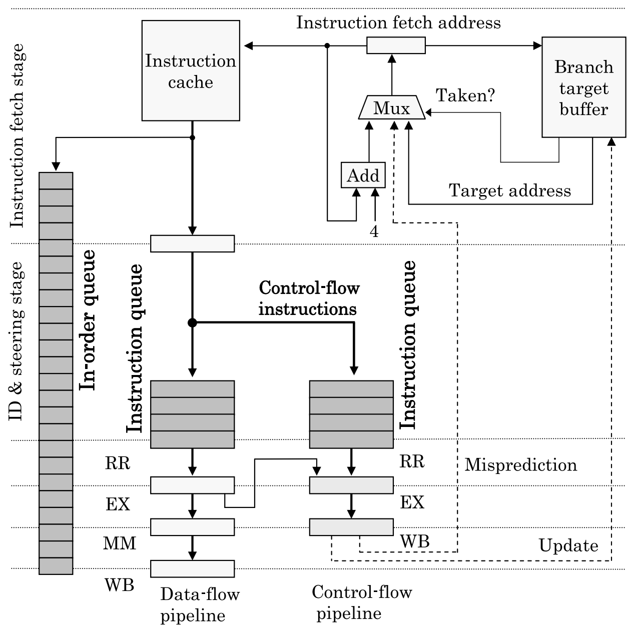

Figure 2: First generation super instruction-flow processor.

In order to classify a control-flow instruction and a data-flow instruction, two instruction queues are inserted between the decode stage and the register read stage. At the decode stage, the fetched instructions are decoded and stored in the proper instruction queue. In the control-flow execution unit, a control-flow instruction is picked out from the instruction queue and processed. Conversely, a data-flow instruction is provided to the data-flow pipeline.

When a branch misprediction occurs, unnecessary instructions must be flushed. The in-order queue is supplemented to realize the pipeline flush. In the fetched order, an instruction tag or identifier is stored in an entry of the in-order queue. The tag is used to identify the instructions fetched after the branch causing the misprediction. The in-order queue is used also to guarantee the order of instruction retirement.

The move instruction, which requires data transfer between the two instruction pipelines, is divided at the steering stage into two instructions called a send instruction and a receive instruction . The send instruction is stored in the instruction queue of the data-flow instructions. The receive instruction is stored in the instruction queue of the control-flow instructions. After the execution of a send instruction, data is transmitted from the data-flow pipeline to the control-flow pipeline on the dedicated datapath.

2.2 Preliminary Evaluation

This section reports the preliminary evaluation results of the first generation super instruction-flow processor. We used a software simulator to model processor with the accuracy at the register-transfer level. In the preliminary evaluation, we eliminate the effect of the Cascade ALU and the data transfer between the data-flow pipeline and the control-flow pipeline. It is the purpose of this section to show the potential of a super instruction-flow processor.

We evaluate the speedup of a first generation super instruction-flow processor compared with the conventional scalar processor. The speedup is summarized in Figure 3. The left end bar is the base performance of the scalar processor with the original code. This code does not contain the overhead instruction for the super instruction-flow architecture. The 2nd bar from the left is the performance of the scalar processor with the code modified for the super instruction-flow architecture. The right four bars are the speedup with the modified code. The right three bars are the speedup of a super instruction-flow processor varying the fetch width (FW) among 1, 2 and 4. The fetch width is the maximum number of fetch instructions per cycle.

From the comparison of the two left bars, we see the 10% speed down by the overhead instructions of the super instruction-flow. From the comparison of the 2nd and 3rd bar from the left, we see that the performance of the scalar processor and a super instruction-flow processor with the narrow fetch width (FW=1) is almost equivalent. On the contrary, the 4th bar from the left, a super instruction-flow processor of FW=2, achieves the 26% speedup for selection sort and the 27% speedup for matrix multiplication. Because a super instruction-flow processor has only two scalar pipelines, the maximum backend throughput is 2. Therefore, even if the fetch width is increased from 2 to 4, there is almost no advantage.

In summary, a super instruction-flow processor of FW=2 achieves the 26% speedup for selection sort and the 27% speedup for matrix multiplication compared with the conventional scalar processor. This is the promising result of the super instruction-flow architecture.

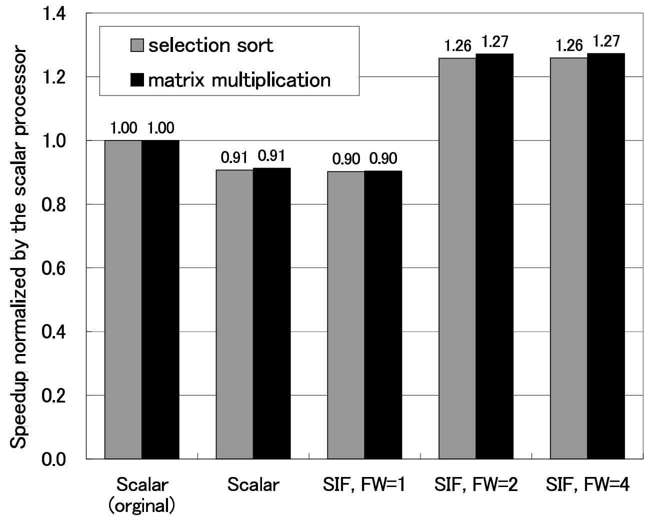

Figure 3: Speedup of a first generation super instruction-flow processor (5-stage instruction pipeline).

3 Discussion and Related Work

The performance of a super instruction-flow processor depends on the quality of the generated executables. In the preliminary evaluation, we used the benchmark programs written in assembly code by hand. As a future work, it is necessary to evaluate the proposed architecture using the large-scale code which is generated by a optimization compiler or a binary translator.

The decoupled access/execute architecture[9] is proposed in 1982. The key concept of the architecture is a high degree of decoupling between the operand access and the execution. The concept of the super instruction-flow architecture is the same but it is a decoupling between the control-flow instructions and data-flow instructions.

In the literature [6], a decoupled front-end architecture is proposed. This architecture allows the branch predictor to run far in advance of the address currently being fetched by the cache. The super instruction-flow architecture focuses on the early resolution of branch instructions not on the deep speculation.

4 Conclusions

Microprocessor performance has improved at about 55% per year for the past three decades. To maintain this performance growth rate, next generation processors with more than one billion transistors must achieve higher levels of parallelism. In this context, a new paradigm or new architecture is required to attain the dramatic boost of available instruction level parallelism.

The goal of this project is to develop a novel processor architecture which mitigates the performance degradation caused by branch instructions. In order to solve this problem, we proposed a super instruction-flow architecture. It has a mechanism to processes multiple instruction flows efficiently. In addition to the architectural proposal, we described the design of its first generation processor.

We reported the preliminary evaluation results of the first generation super instruction-flow processor. Evaluation result show that the super instruction-flow processor with 5-stage pipeline achieves the 26% speedup for selection sort and the 27% speedup for matrix multiplication compared with the conventional scalar processor.

References

[1] A. N. Eden and T. Mudge. The yags branch prediction scheme. In *Proceedings of the 31st annual ACM/IEEE international symposium on Microarchitecture*, pages 69–77. IEEE Computer Society Press, 1998.

[2] Ho-Seop Kim and James E. Smith. An instruction set and microarchitecture for instruction level distributed processing. In *Proceedings of the 29th annual international symposium on Computer architecture*, pages 71–81. IEEE Computer Society, 2002.

[3] J. Lee and A.J.Smiith. Branch prediction strategies and branch-target buffer design. *IEEE Computer*, 17(1):6–22, 1984.

[4] Ramadass Nagarajan, Karthikeyan Sankaralingam, Doug Burger, and Stephen W. Keckler. A design space evaluation of grid processor architectures. In *Proceedings of the 34th annual ACM/IEEE international symposium on Microarchitecture*, pages 40–51. IEEE Computer Society, 2001.

[5] Motokazu Ozawa, Masashi Imai, Yoichiro Ueno, Hiroshi Nakamura, and Takashi Nanya. Performance evaluation of cascade alu architecture for asynchronous super-scalar processors. In *Proceedings of ASYNC-2001*, pages 162–172, 2001.

[6] Glenn Reinman, Brad Calder, and Todd Austin. Optimizations enabled by a decoupled front-end architecture. *IEEE Transactions on Computers*, 50(4):338–355, 2001.

[7] Karthikeyan Sankaralingam, Ramadass Nagarajan, Haiming Liu, Changkyu Kim, Jaehyuk Huh, Doug Burger, Stephen W. Keckler, and Charles R. Moore. Exploiting ilp, tlp, and dlp with the polymorphous trips architecture. In *Proceedings of the 30th annual international symposium on Computer architecture*, pages 422–433. ACM Press, 2003.

[8] James E. Smith. A study of branch prediction strategies. In *Conference proceedings of the eighth annual symposium on Computer Architecture*, pages 135–148, 1981.

[9] James E. Smith. Decoupled access/execute computer architectures. In *Proceedings of the 9th annual symposium on Computer Architecture*, pages 112–119. IEEE Computer Society Press, 1982.