Optimization of Instruction Fetch Mechanisms for High Issue Rates

Thomas M. Conte, Kishore N. Menezes, Patrick M. Mills and Burzin A. Patel

Date of Publication: 22 - 24 June, 1995

Computer Architecture, 1995. Proceedings., 22nd Annual International Symposium on

**Summary:**

The paper proposes solutions to exploit the performance potential of a superscalar processor by enabling the fetch unit to proficiently align and provide instructions at high bandwidth to the decoder and by optimizing compilers.

**Goals, Background and Conclusion:**

With the availability of superscalar processors that are capable of issuing multiple instructions per cycle and have equally capable cores, it is very important for the fetch unit to provide instructions at a matching rate to harness the full potential of the processor. The fetch unit performance can be manipulated by working on the instruction cache performance, branch prediction or instruction alignment. In particular, realigning instructions within the same block which has multiple branches has proved to be a complex task. This paper provides mechanisms to improve the instruction alignment done by fetch unit. Out of the discussed schemes, collapsing buffer tends to hold strong in a wide range of scenarios proving to be a good generic solution which reliably aligns the instructions 90% of the time or more. Combining this with code reordering is suggested as “the best overall solution”.

**Experimental Setup:**

HP 9000/735-class workstations were used with a simplified version of GCC’s intermediate code for the pipeline simulation and 32-bit instruction format. Spike tracing tool was used to capture traces and fed into a processor simulation that assumes a full-Tomasulo, out-of-order execution microarchitecture.

The research is conducted using P14, P18 and P112 microarchitectures, all of which have a scheduling window to resolve dependencies. The cores consist of fixedpoint units (FXUs), floating-point units (FPUs), branch units, and the data cache interface. There is one result bus for every function unit. Messy register file is maintained to execute instructions out of order. PI4 has cache block size 16B, PI8 has size 32B, and PI12 has size 64B. The issue rate for P14 is 32KB, P18 is 64KB, and P112 is 128KB. The branch-target buffer uses a 2-bit counter predictor, is directmapped and has 1024 entries.

The results for the experiment correspond to the following benchmarks: all six SPECint92 benchmarks, mpeg-play, bison, flex and six SPECfp92 benchmarks which were compiled using GCC with the compiler options “-0-f schedule-insn.”

**Critical Enquiry:**

In this paper, the authors come up with a “Method or means of development” and “Design, evaluation or analysis of a particular instance” to enhance the fetch unit’s performance when working with sequential instructions [7]. The authors discuss three schemes viz., Interleaved Sequential, Banked Sequential and Collapsing Buffer. The Interleaved sequential scheme suggests spreading the instruction cache into two banks and prefetching one block beforehand. Two hardware units are used to aid the instruction alignment through this scheme. They are *interchange switch* – used to reverse the order of fetch and target blocks and *valid select logic* – used to select the appropriate instructions from the two banks. The interleaved buffer seems to provide an improvement in performance when compared to the regular sequential execution of instructions but has its shortcomings. One of them is that, it sends the instructions to the decoder but does not remove the unnecessary instructions in between the branch and its target. Another disadvantage with interleaved sequential is that it does not allow fetching across branches.

The Banked sequential scheme overcomes the second disadvantage associated with the interleaved sequential scheme. The hardware setup required for Banked sequential is very similar to interleaved sequential. The only difference is that in Banked sequential, instruction alignment is allowed only when the branch and the target are in different blocks. The third scheme in the paper is the Collapsing Buffer. In addition to inter-branch fetching (the second disadvantage of interleaved buffer), the collapsing buffer tackles with the first disadvantage of interleaved sequential scheme as well, i.e., it gets rid of the unnecessary instructions between the branch and its target before sending them to the decoder. The most efficient way of using collapsing buffer is to implement it as a bus-based crossbar. This implementation allows us to get rid of the two additional hardware required for the interleaved sequential and banked sequential schemes. Compiler optimization by code reordering is another way of improve the performance of the fetch unit as it reduces the number of non-sequential instructions which in turn reduces the rate of the taken branches in the code.

The authors have designed an “Analytic model” to infer their proposal [7]. Based on the results obtained, it is evident that all the three schemes discussed in the paper provide better performance when compared to a regular sequential execution of instructions. When compared amongst themselves, the collapsing buffer scheme shines with a performance much closer to the perfect instructions execution. Also, combination of code reordering with collapsing buffer proves to be the best solution in the given arrangement. As the results are validated by comparing with regular sequential execution (as the baseline), this is nothing but a form of “Analysis” done through a “controlled experiment” [7]. Since the publication of this paper, we have seen many new processors come into picture since then and these schemes should be verified against today’s contemporary processors in order to find if they still provide some performance gain and if it is worth the effort associated with them, when compared to the gain shown with the three machine models used in this paper.

**PART B - Lineage of the Research:**

The research performed in this paper is based on many of its contemporary research advances. For example, the paper intends to provide solution to superscalar processors like [1]. Discussed below are a couple of reference papers that are more relevant to this paper.

When applying the interleaved sequential scheme, where in, before advancing the instructions to the decoder, the non-sequential instructions should be removed from the instructions at the fetch unit. This is achieved by using a branch-transfer buffer in the following way. The BTB is “interleaved by number of instructions in a cache block”. This mechanism is taken from [2], where the authors elaborate on the idea of branch prediction to improve instruction fetch performance. The idea here is that the BTB query should yield the successor block address and predict the eligible instructions in the fetched block for decoding. Using the successor block address, the sequential prefetch block is annulled and the block address and bit-pattern are by using a set of comparators.

Concept of merging, as described in [3], is coalescing instructions from multiple instruction runs. The collapsing buffer scheme implements this idea by removes the unnecessary instructions between a branch and its target. This enables the target to directly follow the branch instruction in the decoder which in turn improves decoder utilization.

The code reordering for optimizing compiler was performed using trace selection and trace layout as explained in [4] which suggests the implementation of a simple global layout process to shift the complexity to the intra-function process which is what the authors are trying to achieve in order to test their schemes.

The ideology and schemes discussed in this paper have been used in many other research works since its publication. In some of the more recent work though, the researchers have tried to come up with alternatives for these schemes. Two such instances are noted below.

Instruction alignment is considered as one of the important aspects of improving fetch unit’s performance. In [5], the authors acknowledge the enhancements suggested at the hardware level by this paper and explore the software domain to identify similar solution that can improve the code alignment performance by monitoring runtime behavior of the program and modifying the code alignment only when necessary. As the implementation moves to the software side with no continuous effort required from the hardware side, this approach reduces the overhead of the scheme(s).

In the contemporary world of computer architecture, it is expected that the design of the superscalar processors should not only exhibit high performance but also reduce the power consumption. In [6], the authors propose techniques to perform the same. In this paper, the authors point out that processors that do not employ aligned access and implement multiple cache banks (as suggested by the schemes discussed in this paper) need to handle more complex tasks like decoding the address multiple times, and will need additional hardware as well. So, such implementation may necessitate more effort from the fetch unit and in turn consume more power.

REFERENCES

1. M. Slater, “AMD’s IC5 designed to outrun Pentium,” Microprocessor Report, Oct. 1994.

2. T. Yeh, “Two-level adaptive branch prediction and instruction fetch mechanisms for high performance superscalar processors”. PhD thesis, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, 1993.

3. M. Johnson, Superscalar processor design. Englewood Cliffs, NJ: Prentice-Hall, 1991.

4. W. W. Hwu and P. P. Chang, “Achieving high instruction cache performance with an optimizing compiler,” in Proc. 16th Ann. International Symposium Computer Architecture, (Jerusalem, Israel), pp. 242-251, May 1989.

5. McDaniel, Michelle, and Kim Hazelwood. "Runtime adaptation: a case for reactive code alignment." Proceedings of the 2nd International Workshop on Adaptive Self-Tuning Computing Systems for the Exaflop Era. ACM, 2012.

6. Gavin, Peter, David Whalley, and Magnus Själander. "Reducing instruction fetch energy in multi-issue processors." ACM Transactions on Architecture and Code Optimization (TACO) 10.4 (2013): 64

7. Shaw, M. 2002. What makes good research in software engineering? Joint European Conferences on Theory and Practice of Software (ETAPS 2002), pp 1-7, April 2002. Int J STTT (2002) 4: 1–7 / Digital Object Identiﬁer (DOI) 10.1007/s10009-002-0083-4