EE482: Advanced Computer Organization Lecture #4
Processor Architecture
Stanford University Monday, 15 May 2000

Fetch Issues

Lecture #4: Monday, 10 April 2000
Discussion Leader:Jinyung Namkoong
Scribe: Paul Wang Lee

1 How to Read a Patent

As an extension to last lecture's "How to Read a Paper", Professor Dally explained the format of patents and gave tips on how to read a patent in preparation for one of the reading materials for next lecture.

- Heading : Includes various information such as the inventors, date of filing, cited references and related publications.
- Terminology : Patents use certain words in certain ways, with different meanings. e.g. 'consists of' has the meaning 'consists solely of', so 'comprises' is used in cases where there may be other parts.
- Claims : The claims at the end of a patent are the ideas that are being claimed as the invention. There are two types of claims, which can be thought of as in a tree structure. Independent claims are central ideas. Dependent claims are claims that elaborate on the central idea.
- "Summary of the Invention" : succinct description of the invention.
- Two requirements of patents : 'Enabling' in that a person of average skill in the art must be able to reproduce the invention based on the disclosed details. 'Best Mode' in that the best configuration and use of the invention known at the time of application must be described.
- Searching through patents : There is a patent search engine at 'www.ibm.com' if one needs to search through patents.

2 Introduction

Present day processors execute multiple instructions per cycle to improve performance. As the number of instructions per cycle increases, it is becoming more difficult to provide the instructions to fill the capactiy of the execution units. Hence, the importance of front-end is increasing. There are four major limiting factors, which are I-cache miss, Branch misprediction, Target Address determination, and Alignment. The three papers discussed today attack different subsets of these problems.

3 Reinman/Austin/Calder Paper [1]

Introduces the Fetch Target Buffer, which is an improved BTB. Claims it is scalable, and has fast decode time.

These front end architectures focus on reducing I-cache miss and latency from target address determination.

3.1 Rationale/General description

- Argument on poor interconnect scaling : with continued scaling, interconnect delays are becoming worse. Memory structures are affected very much by this trend, and thus the front end designs, which contain significant amount of on-chip memory are limited by this effect. Note by Professor Dally : The basic idea is correct, but the wire scaling factors discussed in the paper are wrong. The paper assumes thickness does not scale, but it is scaling now, due to process difficulties, etc.

- The scalability of the proposed architecture comes from the two-level structure of the branch prediction architecture, and the decoupling provided by the fetch target queue. The decoupling moves the slow memory structures out of the critical path.

- Fetch Target Queue: The FTQ is used to bridge the gap between the branch predictor and the instruction cache. The branch predictor produces fetch target block prediction and stores it in the FTQ, and the I-cache consumes the stored predictions. The FTQ permits the branch predictor and I-cache to operate autonomously, and this decoupling allows the latency of large memory in the I-cache to be hidden most of the time.

3.2 Prior Fetch Prediction Architectures and the FTB

Branch Target Buffer BTB's were proposed to provide branch and fetch prediction for wide issue architectures. BTB holds target addresss for branches. Multilevel BTB's were previously found (in 1993) to be not cost effective, but since then, technology has changed.

Basic Block Target Buffer BBTB is indexed by the starting address of the basic block, and holds the taken target address and the fall-through address of the basic block. The branch predictor dictates which address to fetch in the next cycle.

Fetch Target Buffer This is the branch prediction architecture modeled in the paper, and is an improved BBTB, with changes to increase storage efficiency. The fall-through basic blocks and basic blocks with branches that are seldom taken are not stored. In addition, only the lower bits of the fall-through address is stored, since typical fall through addresses are close to the branches.

3.3 Comments

- Relating to Figure 6, it would have been more relevant if axis were in terms of performance.

- The authors seem very obsessed with reducing memory usage. For example, in the FTB, the fall-through address is stored in terms of the lower n-bit address.

4 Luk/Mowry Paper [2]

Prefetching instructions solves the memory latency problem for I-cache misses. This paper discusses an instruction prefetching scheme where the compiler provides hints to the hardware to allow more efficient prefetches.

4.1 Why prefetch?

Memory is slow, so to hide the latency, fetch in advance.

4.2 Terminology/Critria for prefetching

coverage factor : fraction of original cache misses that are prefetched.

unnecessary : if the line is already in the cache.

useless : if it brings a line into the cache which will not be used before it is displaced.

An ideal prefetching scheme would provide 100% coverage and generate no unnecesary or useless prefetches. Timeliness of prefetches are also important, and the prefetching distance should be large enough to hide the miss latency, but not too large so that the line is likely to be displaced by other accesses before it can be used.

4.3 Previous Work

- Next-N-line prefetching : prefetch N sequential lines following the one currently being fetched
- Target-line prefetching : use prediction table to record address of line which most recently followed a given instruction line
- Wrong path prefetching : next N line prefetching combined with always prefetching the target of control transfers with static addresses.
- Markov prefetching : primarily focused on data cache misses, stores correlations of consecutive miss addresses in a miss-address prediction table.

4.4 Cooperative Instruction Prefetching

A fully automatic instruction prefetching scheme proposed by this paper. The compiler and the hardware cooperate to launch prefetches earlier, while maintaining high coverage and reducting the impact of useless prefetches. There are two novel components, *instruction-prefetch instructions*, and *prefetch filtering.*

- Instruction-prefetch instructions : using this instruction, the program provides hints to the hardware. In the implementation proposed by the paper, this instruction is removed in the decode stage to enhance performance.
- Prefetch Filtering : prefetch filters resides between the I-prefetcher and the L2 cache, and reduces the number of useless prefetches. A prefetch bit associated with each line in the I-cache, and two bit saturating counters in the L2 cache are used to decide whether to prefetch or not. Whenever prefetches are gone unused, the counter is incremented, and over a certain threshold, the prefetches are ignored. The counters are reset on fetches. It is used to enable more aggressive prefetching without polluting the cache.

4.5 Comments

Hardware vs. Software-based prefetching Hardware based prefetching is intraprocess, software based prefetching is interprocess. If branching is small, hardware-based is easier, but with many branches, software-based is easier.

5 Conte/Menezes/Mills/Patel Paper [3]

5.1 Alignment

With multiple instructions issuing in each cycle, the fetch unit must extract multiple non-sequential instructions and align them in proper order. This is difficult due to short branches within a cache block, or multiple branches in a cache block.

5.2 Solutions to Alignment Problem

sequential fetch a sequential block, and then apply a mask to extract needed instructions from the block. This is the simplest scheme for fetching multiple instructions per cycle.

interleaved sequential interleave I-cache into two banks, and prefetch one sequential block in advance to allow high issue rates for accesses that span block boundaries. Non-sequential instructions are not allowed.

banked sequential improvement of interleaved sequential, the likely successor address is found for a given fetch address and its successor block is used in the same way as the next block in the previous scheme. Intra-block branches cannot be fetched in same cycle, because the useless instructions in between cannot be eliminated.

collapsing buffer intra-block branches can be taken. *Merging* is achieved, so that the target instruction follows the branch instruction in the decoder. In order to do this, an additional buffer is added to collapse the gaps between valid instructions caused by intra-block branches. There are two possible implementations, a shift register or a bus-based crossbar.

5.3 Comments

- Usage of Harmonic Mean : in order to obtain execution time comparisons, the harmonic mean should be used for speed values. (IPC, etc.)
- The compiler optimizations used for this work is different from the other papers in that it increases I-cache utilization as opposed to fetch efficiency.
- This paper ignores I-cache bandwidth, but bandwidth is highly relevant to performance.
- Proper performance comparison should be done by adding I-cache capacity comparable to the added overhead.

References

[1] Glenn Reinman, Todd Austin, Brad Calder, "A Scalable Front-End Architecture for Fast Instruction Delivery", Proceedings of the *26th International Symposium on Computer Architecture*, May 1999.

[2] Chi-Kung Luk, Todd C. Mowry, "Cooperative Prefetching: compiler and hardware Suport for Effective Instruction Prefetching in modern Processors", Proceedings of the *31st International Symposium on Microarchitecture*, December 1998

[3] Thomas M. Conte, Kishore N. Menezes, Patrick M. Mills, Burzin A. Patel, "Optimization of Instruction Fetch Mechanisms for High Issue Rates", Proceedings of the *22nd International Symposium on Computer Architecture*, June 1995.