Available online at www.sciencedirect.com
SCIENCE DIRECT®
Future Generation Computer Systems 21 (2005) 767–773
FGCS FUTURE GENERATION COMPUTER SYSTEMS
www.elsevier.com/locate/fgcs

The instruction register file micro-architecture

Bernard Goossens*, David Defour

Laboratoire MANO, Université de Perpignan, 52 Avenue Paul Alduy, 66860 Perpignan Cedex, France
Available online 19 November 2004

Abstract

In this paper, we address the issue of feeding future superscalar processor cores with enough instructions. Hardware techniques targeting an increase in the instruction fetch bandwidth have been proposed such as the trace cache microarchitecture. We present a microarchitecture solution based on a register file holding basic blocks of instructions. This solution places the instruction memory hierarchy out of the cycle determining path. We call our approach, instruction register file (IRF). We estimate our approach with a SimpleScalar based simulator run on the Mediabench benchmark suite and compare to the trace cache performance on the same benchmarks. We show that on this benchmark suite, an IRF-based processor fetching up to three basic blocks per cycle outperforms a trace-cache-based processor fetching 16 instructions long traces by 25% on the average.

Keywords: Trace cache; Instruction memory; Register file; Fetch stage

1. Introduction

Fetch bandwidth is an essential issue in actual processors. Two factors tend to limit the processor fetch capacity per cycle:

1. The presence of control flow instructions breaking the sequential nature of fetching.
2. The reduction of the clock cycle width that leads to an increased latency of the fetch path and therefore of the miss-predict penalty.

To overcome the first factor, compiler [4,6] and micro-architectural [1,10,15] solutions were proposed. Among them, the trace cache [17,23] is the leading micro-architectural advance. Evidence is its use by industry architects within the Pentium 4 [20].

Concerning the second limitation in the fetch bandwidth due to the cycle width reduction, some new proposals are arising. The fetch target buffer (FTB) [21] is a multi-level branch target buffer (BTB) [18] decoupled from the fetch path to allow a fast prediction. However, the instruction cache remains in the fetch critical path and today yet must be pipelined to scale with the cycle (today, a fetch has a two cycles latency). Prefetching techniques have also been proposed to hide instruction cache miss latency [5,19,22,24].

In this paper, we propose an alternative to both the trace cache and the FTB that we called instruction register file (IRF) machine. Section 2 presents the trace cache micro-architecture and some of its drawbacks. Section 3 describes the proposed architecture. Section 4 is devoted to the performance measures of the IRF and

* Corresponding author.
E-mail address: goossens@univ-perp.fr (B. Goossens).

0167-739X/$ – see front matter © 2004 Published by Elsevier B.V.
doi:10.1016/j.future.2004.05.017

compare it's results with the trace cache (TC). Section 5 concludes this paper.

2. Overview of the trace cache micro-architecture

The purpose of the *trace cache* depicted in Fig. 1 is to feed the execution pipeline with enough instructions, by removing uncertainty due to control flow instructions. To achieve it, the trace cache holds dynamic instruction sequences, including taken branches. These instruction sequences use block of instructions called *Basic Block* or BB. A basic block in the context of the trace cache is defined by the three following rules:

1. A BB has at most one control flow instruction.
2. If a BB has a control flow instruction, it is the ending one.
3. A BB has at most n consecutive instructions where n is the trace cache block size.

To be executed, full BBs are concatenated together to form a *trace line*. The trace line corresponding to the program counter (PC) is read during the fetch stage from the *trace cache*. Then, the fetched line is cut at the longest prefix according to the trace path predictor outcome and delivers this prefix to the dispatch stage: this is known as "partial matching" [23]. In case of a trace cache miss, instructions are taken from the memory hierarchy.

These instructions are sent "in-order" to the reservation stations during the dispatch stage. In the same time, these instructions are also sent to a trace line buffer, which is cached when full. Therefore, multiple BBs may take place in this buffer. It should be noticed that in some implementation the fill unit is placed in the commit stage or uses trace preconstruction [12].

The fill unit integrates a new BB to the buffer containing the trace line under construction if all of the following conditions hold:

- The BB fits in the buffer.
  This conditions serves to limit the redundancy in the trace cache.
- The buffer does not contain any indirect jump or trap instruction.
  Indirect jumps may be followed in the run trace by different BBs, which would lead to multiple trace lines with a common prefix.
- The buffer contains at most q conditional branch ended BBs.
  The value of q is the number of branch predictions that the multiple branch predictor can predict in a single cycle.

At the other end of the pipeline, at control flow instruction retire time, the processor core may correct a miss-predicted path by sending a correcting address to PC. It also adjusts the trace path predictor according to the control flow instruction computed direction and target.

2.1. Drawback of the trace cache

2.1.1. Path associativity

The third rule defining a BB ensures that it always fits in a trace cache line. Therefore $m, m > n$ consecutive instructions are split into several BB even though no control flow instruction requires it. So, a BB is either ended by a control flow instruction or has length n. A consequence is that two BBs may share a common suffix, but two different BBs cannot start with the

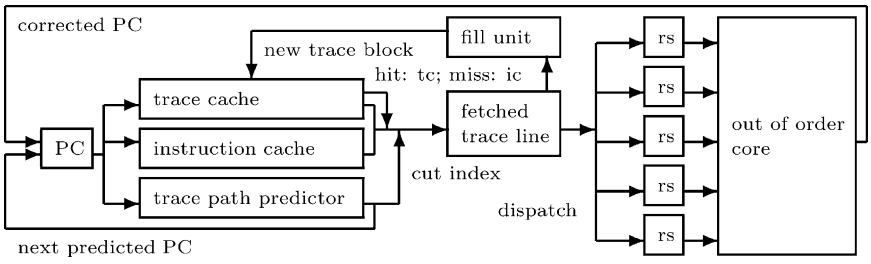

Fig. 1. The trace cache micro-architecture.