HY425

Machine Assignment 2

Assignment: 1/12/2008

Due Date: 15/12/2008

Instructions: For this assignment you will need to submit a write-up (in .pdf) with your experimental findings from the SimpleScalar simulator. Send your solution (in compressed tar format) to Stamatis Kavadias (kavadias@ics.forth.gr). Use the following subject in your e-mail (please do not change the subject!): **HY425: MA2 Submission.**

Start early!

Introduction

Your goal in this assignment is to implement two simple instruction and data prefetching mechanism in the SimpleScalar out-of-order simulator. You will implement and evaluate two sequential instruction/data prefetching schemes using the same benchmarks that you used in the first machine assignment (anagram, go, compress, gcc).

Preparation

Study thoroughly the files cache.c, cache.h, and sim-outorder.c. Study very carefully the execution path of the out-of-order processor simulator for data cache accesses and instruction cache accesses, both hits and misses. A thorough understanding of the cache access execution paths (the cache_access function and the timing of cache accesses implemented in sim-outorder.c) will help you in your implementation. You will need to modify the cache access functions in the out-of-order simulator to implement the prefetching schemes listed in this assignment. Certain data and instruction cache accesses will trigger prefetching operations. Prefetching operations should be almost identical to regular data access operations, with the exception that they are triggered transparently and implicitly from other load/store/instruction fetch operations issued by the processor. You are asked to implement the same prefetching mechanism in both the L1 instruction cache and the L1 data cache of the processor.

Part A: Look-ahead prefetching on a miss (50 points)

The first prefetching scheme that you are asked to implement works as follows: If the cache has a cache block size B, when the processor accesses a word in address A, and the processor misses in the L1 cache, the processor will start fetching the block containing A from the lower level of the memory hierarchy and simultaneously (at the same time) start prefetching the block starting at address (A & ~(B-1)) + B from the lower level of the memory hierarchy. Notice that prefetching should seek the block in the lower levels of the memory hierarchy from the faster to the slower (i.e. first in the L2 cache, then in main memory).

The previously described strategy is also known as *prefetch-on-miss* and *one-block lookahead prefetching*. Assume that the processor has all the necessary hardware resources for issuing two simultaneous accesses to all levels of the memory hierarchy below the L1 cache (L2, memory)[1]. In all cases, prefetching should proceed in parallel with processor computation (i.e. prefetching and instructions executed by the processor should overlap). It is very important to keep track of the timing of the prefetching and the timing of the processor operations properly and separately! If the prefetched data arrives in the cache after the processor issues a request for this data (through a load, store, or instruction fetch operation), then the memory access is a cache miss. If the prefetched data arrives in the cache before the processor issues a request to this data, then the request is a hit. Simplescalar's cache data structures permit you to track when a requested cache block will be available.

You are asked to implement a prefetching mechanism, where the prefetched data is placed in the L1 (instruction or data, depending on the type of access) cache of the processor. Note that this means that prefetched data may (and will) replace useful data that is already in the L1 cache. Note also that the block you are trying to prefetch may already be in the L1 cache (in which case you don't need to refetch the block from another layer of the memory hierarchy). Alternatively, the block may already be in the L2 cache, in which case the prefetching operation will take less than when prefetching is done from main memory.

You will use the default parameters of SimpleScalar's out-of-order simulator for this study. You are asked to evaluate your prefertcher by taking the following measurements:

i) Measure prefetch efficiency. Prefetch efficiency is defined as the ratio of the number of prefetched blocks that are referenced by the processor and the references to these blocks hit in the L1 cache (meaning that the prefetching of these blocks was effective), over the total number of blocks prefetched by the processor in the cache. Measure the efficiency separately for the instruction and the data cache.

ii) Measure L1 cache miss ratios, with and without your data prefetching mechanism activated. Measure the ratios separately for the instruction and the data cache.

iii) Measure performance in IPC (instructions per cycle) with and without your data prefetching mechanism activated. Measure the impact of prefetching both instructions and data (i.e. do not separate the two in these measurements).

Part B: Tagged Data Prefetching (50 points)

In Part B, you need to modify your sequential prefetching strategy as follows. Every time the processor references a block and misses in the cache, the processor prefetches the next sequential block (as in Part A). When a prefetched

[1] This would require the implementation of additional hardware support in SimpleScalar, which we are not asking for in this assignment.

block is accessed by the processor and the block is in the L1 cache, then the next sequential block is also prefetched. To accomplish this, you need to add a tag to each block that you fetch in the cache. If the block is fetched in the cache by a prefetching operation (not a regular load/store), the block is marked by setting its tag to 1. When the processor accesses the prefetched block, the tag of this block is reset to 0, and the processor initiates a prefetch operation for the next sequential block. Tagged prefetching has shown to be more effective than prefetching on a miss.

Repeat the measurements you did for Part A.

Final Remarks

You will notice that your version of SimpleScalar assumes zero-cycle writes in the cache and unlimited write buffers for write operations. You will follow the same assumption, i.e. your implementation will only concern instruction/data read operations, and the corresponding miss and prefetch handling.

You are asked to submit a report detailing your implementation of data prefetching in SimpleScalar and your findings. Use your favourite editor to prepare the documentation and charts, but make sure that everything can be converted to .pdf for your submission.