INF2270, performance degradation due to pipelining hazards, cache misses and page failures

P. Häfliger

March 23, 2011

Abstract

In this exercise you will try to gauge the dependence of computer performance on cache hit rate and page failure rate, and the dependence of pipeline speedup on (control) hazards.

Cache Miss and Page Failure

1. Imagine you have implemented a program that will execute 1000 instructions. Assume that each of these instructions that do not access the memory require one clock cycle to execute and that the clock frequency is 3GHz. 200 of these instructions, however, will access the memory. Let's assume that there is no penalty at all for a cache access and that these instructions also just need one clock cycle to execute. If the program is made well or the programmer is lucky, all of the memory accesses will be cache hits (even the first memory access, since the block is already in the cache prior to execution). How long will the program take to execute?

2. Let's be more realistic and assume that the cache hit rate is 95%. Let us further assume that a cache miss will cause a memory access to last 20 clock cycles. How much time will the program need to execute? (Do not assume any fancy compiler level pipeline tricks that allow some instructions to be executed while waiting for the memory access to complete! Assume that execution stalls until each memory access is complete, i.e. the full 20 clock cycle penalty will be felt.)

3. Let us be more nasty still and assume that at the start of the program the data is stored on a virtual memory page that is not at present in the memory at all and needs to be loaded first (or simply assume that the program needs to be loaded from the hard drive into the memory for execution). The penalty for the page failure is 1 million clock cycles. How long, before execution is complete?

4. So what would be your comment on the usefulness of optimizing the number of instructions for very short programs? How would your statement

change if this short program is part of a bigger program and will be executed many times over in the course of the execution of the big program?

Pipeline Speedup

1. The speedup of a pipelined program is the ratio between execution time of a sequential execution and pipelined execution of the program. In the lecture we have seen that the 'filling' of the pipeline causes some initial delay, i.e. an 'offset' in the execution time that pipelining cannot get rid of. Thus, if a program needs executes n instructions on a processor with k pipelining stages and each stage is executed within one clock cycle period of duration t, the speedup is never exactly equal to k but somewhat smaller. What's the speedup when considering the penalty for illing' the pipeline, given that n=100, k=5, and t=1ns?

2. Pipeline hazards are another cause for the speedup to fall short of the theoretical optimum k. Let us try to estimate the penalty of, let's say, control hazards in a CPU with no branch prediction. Let's say that the probability of a branch instruction in a certain program is P_b=20% and the probability of it being taken is P_t=70%. Let us further assume that a branch not being taken causes no penalty but a branch being taken causes the branch instruction to cost 3 clock cycles instead of just 1. In a program that executes 1000 instructions what is the speedup that pipelining achieves in this scenario? For the entire program, what is the average number of clock cycles needed per instruction (CPI). (Ignore the penalty for filling up the pipeline: it's relatively small for a 1000 instruction program)

3. How do these numbers improve if branch prediction is included and predicts correctly in P_c=70% of the executed branch instructions? Assume that a correctly predicted branch causes no penalty and an incorrect prediction again causes the branch to use 3 clock cycles instead of 1.