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

P. Häfliger

April 6, 2010

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. One clock cycle t of a clock with a frequency f of 3GHz lasts:

   $t = \frac{1}{f} = 0.33\text{ns}$

   With no cache misses and assuming one clock cycle per instruction and no penalty for a cache access (realistic for L1 cache in some architectures) the average number of clock cycles per instruction CPI is 1. A program of n instructions when n=1000 will take:

   $tn = 0.33\text{ns} \times 1000 = 0.33\mu\text{s}$

2. With a hit rate of 95% and 200 memory accesses out of 1000 instructions, 10 instructions will cause a penalty of 19 clock cycles. In other words the execution of the program wil take 1190 clock cycles, i.e. 19% longer because of only 5% cache miss rate among only 20% of the instructions that access the memory. A quite severe performance degradation despite a rather good cache hit rate.

   Execution time becomes:

   $800t + 190t + 10 \times 20t = 1190t = 0.39\mu\text{s}$

   Average clock cycles per instruction becomes:

   $CPI = \frac{1190}{1000} = 1.19$

3. considering the load time for this program from the hard drive or just assuming a page failure in the virtual memory for the first data access will cause a 1000000 clock cycle penalty in addition, which simply will dominate the total execution time of now 1001190clock cycles, i.e. 0.33ms

4. Thus, optimization of small programs does not really pay off. However, optimization of small functions/procedures/routines that are executed frequently (e.g. in a loop) within a large program that remains in memory longer than the time it takes to load it into memory is very much worth your while.

Pipeline Speedup

1. Ideally one would expect a speedup of k since a non-pipelined program takes nkt seconds to execute and a pipelined program may approach an execution time of nt seconds. Thus, the speedup would be:

   $\frac{nkt}{nt} = k$

   However, with the initial delay of 'filling' the pipeline, the first instruction does not profit from pipelining and needs k clock cycles to finish. Only the following instructions will cause only a single clock cycle additional delay per instruction. Thus, the execution time is

   $kt + (n-1)t = (k+n-1)t$

   and the speedup

   $\frac{nkt}{(k+n-1)t} = \frac{nk}{k+n-1}$

   which only approximates k for very large numbers of instructions n. In our specific example the speedup is thus

   $\frac{100\times 5}{5+100-1} = \frac{500}{104} = 4.81$

   Average Clock cycles per instruction here is

   $CPI = \frac{104}{100} = 1.04$

   Since programs usually are quite a bit longer than just 100 instructions, this penalty is usually not severe and mostly not worth mentioning.

2. Pipeline hazards are more severe performance limiting factors, and control hazards are most often the most serious (although, one can easily construct programs that suffer more from other types of hazards). In this example the CPI is

   $(1 - P_bP_t) \times 1 + P_bP_t \times 3 = 1 + 2P_bP_t = 1 + 0.14 \times 2 = 1.28$

   Execution time, thus, becomes

   $nt * \text{CPI} = 0.42\mu\text{s}$

   (that's another convenient formula for the execution time if yo happen to know the CPI). Note that with no branch prediction the situation is often as bad as depicted in this example, since P_t is often quite high.

3. With branch prediction, a significant improvement is often achieved. In our example the CPI becomes:

   $CPI = 1 - P_b(1-P_c) + 3P_b(1-P_c) = 1 + 2P_b(1-P_c) = 1.12$

and the execution time

$nt * \mathrm{CPI} = 0.37\mu\mathrm{s}$