Instruction Prefetching

Ritesh Singh

October 2019

1 What is Instruction Prefetch?

In computer architecture, instruction prefetch is a technique used in microprocessors to speed up the execution of a program by reducing wait states. Modern microprocessors are much faster than the memory where the program is kept, meaning that the program's instructions cannot be read fast enough to keep the microprocessor busy. Adding a cache can provide faster access to needed instructions. Prefetching occurs when a processor requests an instruction from main memory before it is actually needed. Once the instruction comes back from memory, it is placed in a cache. When an instruction is actually needed, the instruction can be accessed much more quickly from the cache than if it had to make a request from memory. Since programs are generally executed sequentially, performance is likely to be best when instructions are prefetched in program order. Alternatively, the prefetch may be part of a complex branch prediction algorithm, where the processor tries to anticipate the result of a calculation and fetch the right instructions in advance. In the case of dedicated hardware the prefetch can take advantage of the spatial coherence usually found in the texture mapping process. In this case, the prefetched data are not instructions, but texture elements that are candidates to be mapped on a polygon.

2 Why do we need Prefetching?

- We can incur a large performance penalty if there is a miss in the i- cache
- For the next 10-50 cycles, there will be no instructions to fetch if there is an L2 hit
- If there is a miss in the L2, we have nothing to do for hundreds of cycles
- The pipeline will have bubbles (idle cycles), IPC will suffer

To avoid the above scenario,we **Prefetch** the memory addresses.That is we **Predict** the memory addresses that will be accessed in the future. Fetch them from the lower levels of the memory hierarchy before they are required.

3 Some Prefetching Schemes

- Long Cache Lines
- Next Line Prefetching
- Target Line Prefetching
- Hybrid Scheme
- Wrong Path Prefetching

4 Long Cache Lines

This is the simplest from of prefetching. When a instruction cache miss occurs, more than one instruction is brought into the cache as a (long) cache line. Thus, probability that the next instruction needed is in the cache increases. This in turn results in a reduction in the number of cache misses. However this method increases memory traffic as well as cache pollution. Cache pollution increases because many lines may only be accessed partially before it is displaced.
Choice of the length of cache lines depends on the locality and sharing property of programs as well as available memory bandwidth. Programs with good spatial locality usually benefits from using longer cache lines as most of the data in the cache line is likely to used before it is invalidated. In addition to the properties of the program, length of the cache line is also determined by the available memory bandwidth. This is because as length of cache line increases, the width of the memory bus also has to increase.

5 Next Line Prefetching

the cache line that is next to the current cache line (the cache line containing the current instruction) is prefetched automatically if it is not already in the cache.
The advantage of this method is that it is simple to implement, not a lot of additional logic is required. Since we know which cache line the current instruction is in, finding the next cache line is trivial and performance is fairly good if branches frequently execute the fall through path.
But it is not very useful in the case where branch is taken. In unconditional jumps and procedure calls where the branch is always taken, next line prefetching can cause increase in memory traffic and cache pollution as it is not likely that the prefetched cache lines are going to be used.
Next line prefetching has been shown to reduce cache misses by **20-50 percent**. Due to its ease of implementation and small cost, the next line prefetching scheme can be found in many microprocessors.

6 Target Line Prefetching

In target line prefetching, the processor always tries to prefetch on the correct path. This means that in the sequential portion of the code, the next cache line is prefetched as in next line prefetching. However if there are control instructions such as a conditional branch or jump, the target of the control instruction is prefetched.
Being able to prefetch the target of a control instruction means that the effective of address of the control instruction has to be known, even before the branch instruction is executed. To do this a target prefetch table is maintained by the processor. Furthermore, an initial execution of the program is required to create the initial entries in the prefetch table. Each entry in the target prefetch table contains the following pair of information:tag to current line, tag to next line
When execution transfers from one cache line to another, two things happen. First, the tag to next line field in the previous entry of the target prefetch table is updated with the current cache line's tag. Second, a search is done to determine if the next line of the current entry in the prefetch table is in the cache. A fetch is initiated if the cache line is not found. Therefore, if we were to follow through the current line, next line link in the target prefetch table, it will trace out the logical path of the program, assuming that the table is large enough.
The advantage is that it provides good performance if the execution flow of the program follows the path of the previous execution.
Some disadvantages of Target Line Prefetching-

- Very costly in terms of chip area and added complexity as the prefetch target table and the associated update and search logic is large and complex.
- Requires an initial execution of the software to establish the initial table.
- If the successive execution of the program does not follow the same logical execution path as it's previous execution, target line prefetching will cause more harm than good.
- Increases memory traffic and cache pollution due to the above reasons.

7 Hybrid Scheme

This is a combination of next line and target line prefetching. In this scheme, both the next line and the target of a branch are prefetched. This provides double protection, and the cost in terms of hardware can actually be lower than target line prefetching. The next line part functions as the next line prefetching as mentioned above.
The Target part, however, is slightly different. A target prefetch table is maintained as in target line prefetching. If the successor line is the next sequential cache line, it is not added to the target buffer as it will be taken care of by

the next line part, thus saving buffer space. Therefore, in hybrid prefetching the prefetch target buffer can be smaller than in target prefetching to produce similar effects.
The advantages are-

- The performance is shown to be approximately the sum of the gain achieved by next line and target line prefetching.
- Buffer size can be smaller than target line prefetching for comparable performance.

The disadvantages are-

- Very costly in terms of chip area and added complexity as the prefetch target table and the associated update and search logic is large and complex.
- Increases memory traffic and cache pollution.

8 Wrong Path Prefetching

Wrong path prefetching was introduced by Pierce and Mudge . Wrong path prefetching is fundamentally very different from any of the schemes mentioned above. Long cache lines, next line, target line and hybrid schemes all tries to prefetch instructions that are in the correct execution path of the program. However, wrong path prefetching prefetches on the simplest wrong path. It prefetches target lines that are not taken. This is done in hopes that mispredicted instructions brought into the buffer during speculative execution will turn out to be prefetched for later correctly predicted instructions.

Similar to the hybrid scheme, wrong path prefetching combines both next line and target prefetching. The next line prefetching portion is similar to scheme Next Line Prefetching. For the target line portion, unlike target line prefetching described earlier, no target line address information is saved and no attempt is made to prefetch only in the correct execution path. Instead the line containing the target of the conditional branch is prefetched. Thus, both paths of the conditional branch are always prefetched. At the first look, it would seems that hybrid and wrong path prefetching are similar as they both prefetch both paths of the conditional branch. However, hybrid prefetching prefetches instructions that will be executed almost immediately with the help of a prefetch history table. Wrong path prefetching on the other hand prefetches instructions that will not be executed immediately and no history information is needed.

However, in a typical pipeline, because target of a branch is computed at such a late stage, prefetching the target when the branch is taken is unproductive. A cache miss and a prefetch request will be generated at the same time. Thus, the target prefetch portion of wrong path prefetching can only perform

potentially useful prefetch if the branch is not taken. It is hoped that if execution returns to the branch in the near future and the branch is taken, the target line will already be in the cache from the previous prefetch.

The advantages are-

- No or very little extra hardware is required. The next line portion requires minimal additional hardware, and the address from the target portion is calculated by the existing instruction decoder.
- Performs better if there is a large difference between CPU and memory speed. This is because all other schemes described earlier all prefetch instructions that will be executed almost immediately and if memory is slow, prefetch may not be initiated soon enough, However in wrong path prefetching, prefetches are down a path which is not immediately taken, thus there is more time to prefetch the line from slow memory.

The disadvantages are-

- Increase in memory traffic and cache pollution as lines may be prefetched down the wrong path that will never be executed.

9 Comparison between different Methods

Results stated in the following two sections are obtained from the paper titled **"Wrong-Path Instruction Prefetching" by Pierce and Mudge**.
Figures 1 and 2 below compare the penalty cycles and memory traffic per thousand instructions of the next line, hybrid, wrong path 1,2,3 prefetching schemes as well as with no prefetch. From the figures below, we can see that wrong path prefetching performs better than all the other prefetching schemes and the performances is much better compared to no prefetching. However, with the increase in performance, the amount of memory traffic also increases as shown in figure 2.

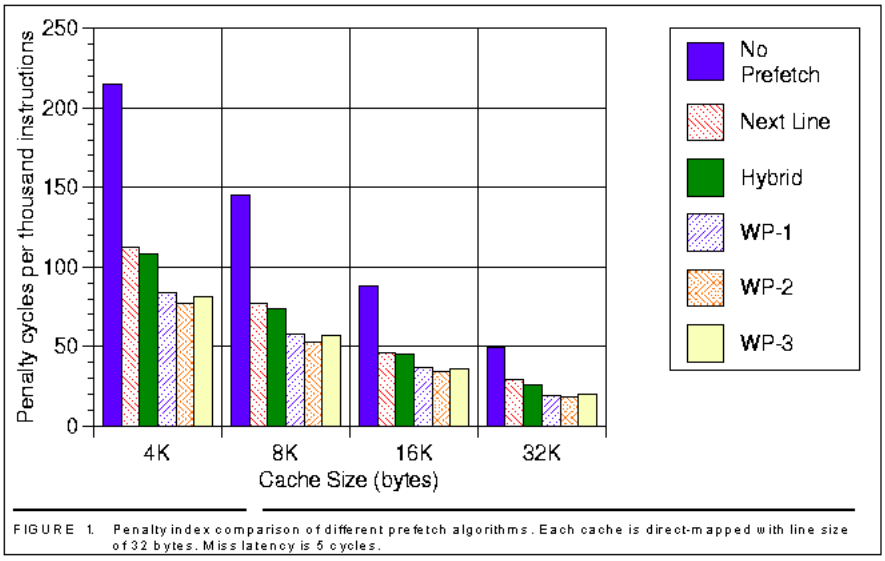

Figure 1: Penality Index

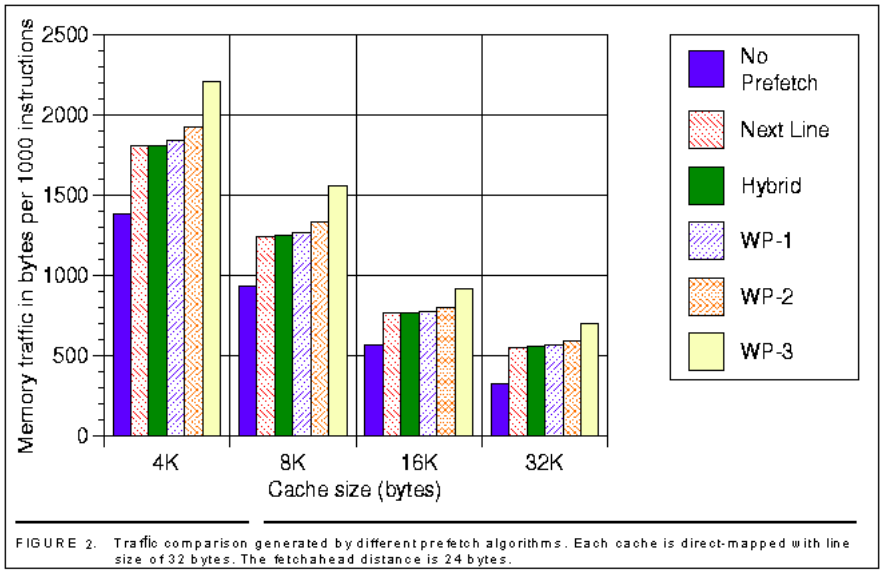

Figure 2: Traffic Comparison