A Mathematical Model of Hardware Prefetching

Kevin Dick
California Institute of Technology
kdick@caltech.edu [*]

Abstract

Modern processors are sometimes equipped with hardware prefetchers, which attempt to eliminate cache misses by prefetching data from main memory into the cache before it's explicitly requested. The practical improvement allowed by prefetching is studied exhaustively for various algorithms in [4]. Prefetching is observed to give substantial speedup for traditional and even specially designed cache efficient algorithms. These experimental results demand a more thoroughly developed theoretical framework for analysis of the prefetching phenomenon. In this paper, an analytical model of the hardware prefetcher is introduced. The model is compared to the existing experimental results, to which it has reasonable agreement. Finally, implications of the model for future development of algorithms are discussed.

[*]Supervised by Richard E. Ladner, University of Washington. ladner@cs.washington.edu.

The time complexity of algorithms has traditionally been studied exclusively in terms of the instruction count. More recently, efforts to construct cache efficient algorithms have focused on adapting asymptotically optimal algorithms to the multiple tiers of the memory hierarchy. Frigo *et al.* present several algorithms which are oblivious to hardware parameters with an asymptotically minimal number of instructions and cache misses under a simple two-tiered memory model [1]. An additional hardware feature, though, makes several theoretically optimal cache efficient algorithms less effective in practice. The *hardware prefetcher* of the Intel Pentium 4 processor allows for bytes to be loaded from main memory into the L2 cache before being explicitly requested. To anticipate desired bytes, the prefetcher keeps track of previous cache misses. Its range is limited to 256 bytes ahead of current accesses and it can operate on up to eight concurrent data streams [2].

Given the magnitude of its allowed speedups, surprisingly little attention has been given to studying the effects of the prefetcher for various algorithms. In the experiments by Pan *et al.*, standard and cache efficient implementations of algorithms are studied with and without prefetching enabled. The prefetcher almost universally allows for significant improvement. For some of these algorithms, the prefetcher makes standard implementations faster than cache efficient ones, even when the cache efficient methods outperform standard ones without the prefetcher. This disparity implies that speedups delivered from the prefetcher are highly sensitive to memory access patterns. In order to understand these distinctions, a mathematical model of the prefetcher is developed.

Despite being a hardware mechanism, the prefetcher lends itself readily to analytical study. We begin by defining several relevant parameters. For simplicity, assume that we have main memory and a single cache level. Our cache will have blocks of size b and require c processor cycles to pull up a block from main memory on a cache miss. Suppose we fix some stride s and choose to access every s bytes in memory. We'll consider the number of processor cycles spent per byte accessed. For each access, we spend a total of p cycles putting the byte into a register and performing some sort of computation, then possibly writing to the byte. With the prefetcher disabled, counting the time needed for our operations gives

$$\text{Cycles per memory access} \quad = \quad \begin{cases} c+p & \text{if } s > b \\ \frac{s}{b}\cdot c + p & \text{if } s \leq b \end{cases} .$$

Enabling the prefetcher, we require an additional parameter r denoting its range. If the time spent computing with the current block is enough to allow the prefetcher to bring in our next block entirely, cache misses are effectively removed. Otherwise, we need to finish pulling the next block into the cache after finishing with the current block. We assume that the prefetcher range r exceeds the block size b. Counting up the number of cycles for our different cases, we get

$$\text{Cycles per memory access} \quad = \quad \begin{cases} c+p & \text{if } s > r \\ c & \text{if } b < s \leq r \text{ and } c > p \\ p & \text{if } b < s \leq r \text{ and } c \leq p \\ \frac{s}{b}\cdot c & \text{if } s \leq b \text{ and } c > \frac{b}{s}\cdot p \\ p & \text{if } s \leq b \text{ and } c \leq \frac{b}{s}\cdot p \end{cases} .$$

We can compare these analytical predictions to the results of [4]. The setup there used blocks of 64 bytes and had a prefetcher range of 256 bytes. The cost c of a cache miss and the processing parameter p are estimated at 100 and 20 cycles, respectively. The experiments of [4] also used the adjacent cache line prefetch feature, which effectively considers adjacent pairs of cache lines as single blocks. This is equivalent to replacing b with $2b$ in the model. Then the predictions of the described model give

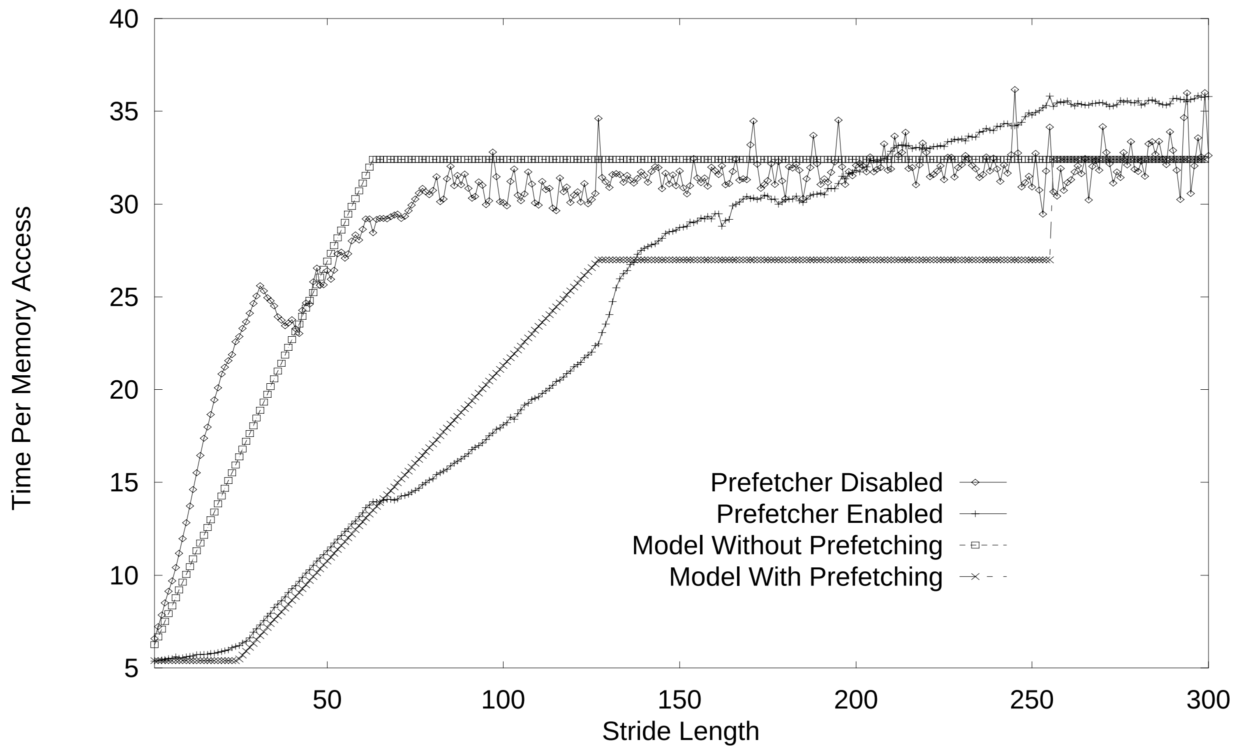

Figure 1: Comparison of experimental results with model predictions. The stride is given in bytes and the units of time are only relative.

reasonable agreement with the experimental results in Figure 1. In particular, the plots of both without the prefetcher show a roughly linear increase until the stride surpasses the block size, at which point the curves flatten. With the prefetcher enabled, we have a flat plot when the prefetcher can eliminate the cost of cache misses entirely. As the cost of cache misses becomes significant the cycles per access scale linearly. Finally, when the prefetcher is out of range we reduce to the original case. The agreement between this model and the experimental results is limited, due in part to the model's false assumption of a single level cache. Although more intricate experiments are feasible, the prefetcher's functionality depends critically on linear memory access patterns, hinting at the difficulty of modeling experiments with less patterned accesses. Future work might demand additional details on the prefetcher's ability to predict cache misses, which could be used to construct a more sophisticated model in conjunction with further experiments.

This theoretical model, while substantially simplifying the prefetcher's functionality, allows for some basic qualitative observations useful in algorithm design. Generally, long sequences with fixed strides of memory accesses are desirable. As the model suggests, the prefetcher is most effective at eliminating the cost of cache misses with large p and small s parameters. This translates into allowing enough time for the prefetcher to pull in the next cache block while useful computation is still being performed. As p shrinks and s increases, the prefetcher is less capable of concealing cache misses. In general, then, construction of algorithms designed to perform well under prefetching relies critically on using bytes close together in a linear fashion. Performing light computation after each successive memory access as opposed to heavy computation after several accesses is preferred. These observations justify the successes of the algorithms studied experimentally and imply guidelines for algorithm design in the context of prefetching.

References

[1] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. *Cache-Oblivious Algorithms.* 40th Annual Symposium on Foundations of Computer Science (FOCS). pages 285-298. 1999.

[2] G. Hinton, D. Sager, M. Upton, D. Boggs, D. Carmean, A. Kyker, and P. Roussel. *The Microarchitecture of the Pentium 4 Processor.* http://www.intel.com/technology/itj/q12001/pdf/art_2.pdf.

[3] *Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 3B.* http://www.intel.com/design/processor/manuals/253669.pdf.

[4] S. Pan, C. Cherng, K. Dick, and R. E. Ladner. *Algorithms to Take Advantage of Hardware Prefetching.* To appear in the Workshop on Algorithm Engineering and Experiments (ALENEX). 2007.