Stride Directed Prefetching Using SPT

Yusheng Weijiang, Yuhui Zheng

The goal of this project was to implement a stride-based prefetcher as outlined in the paper "Stride Directed Prefetching in Scalar Processors" by John W. C. Fu, Janak Patel, and Bob Janssens. The implementation given in the paper involves using something called a Stride Prefetch Table (SPT) in order to calculate prefetches for the cache. The SPT keeps track of every instruction address of each executed memory instruction, as well as the address accessed during each of these instructions. When an instruction with the same address is accessed more than once, we know that the program is running in a loop. Within this loop, if the memory address that is accessed changes, this is a "stride," meaning that the program is probably loading or storing to memory locations in a sequence. Thus, when a stride is detected, we can prefetch the next memory location in a sequence so we don't have to go through the process of fetching it to cache the next time it is used.

Due to the date of the paper's publication, many things were simulated differently in our tests. In the paper, the cache used is a single unified L1 cache, with total size of only 4 KB. Our experiment used a 65 KB data cache, a 32 KB instruction cache, and a 2 MB L2 cache. In addition, the paper tested various types of prefetching on hit/miss, while we tested either no prefetching at all or prefetch on every instruction. The benchmarks that we ran were spec2k6, as the ones used in the paper are very old. Lastly, the paper mentions a few tests done with finite size SPTs, which would only hold a certain number of addresses and have to kick out unused ones when it filled up. We used an infinite size SPT, as was tested for most of the paper, and didn't run any tests on the finite size SPT.

A list of the spec2k6 benchmarks used is as follows, divided up into integer data and floating point data. Generally, prefetching should work better on floating point benchmarks, as they access more sequential cache data. Graphs of cache miss rates across all the benchmarks are shown starting on the next page.

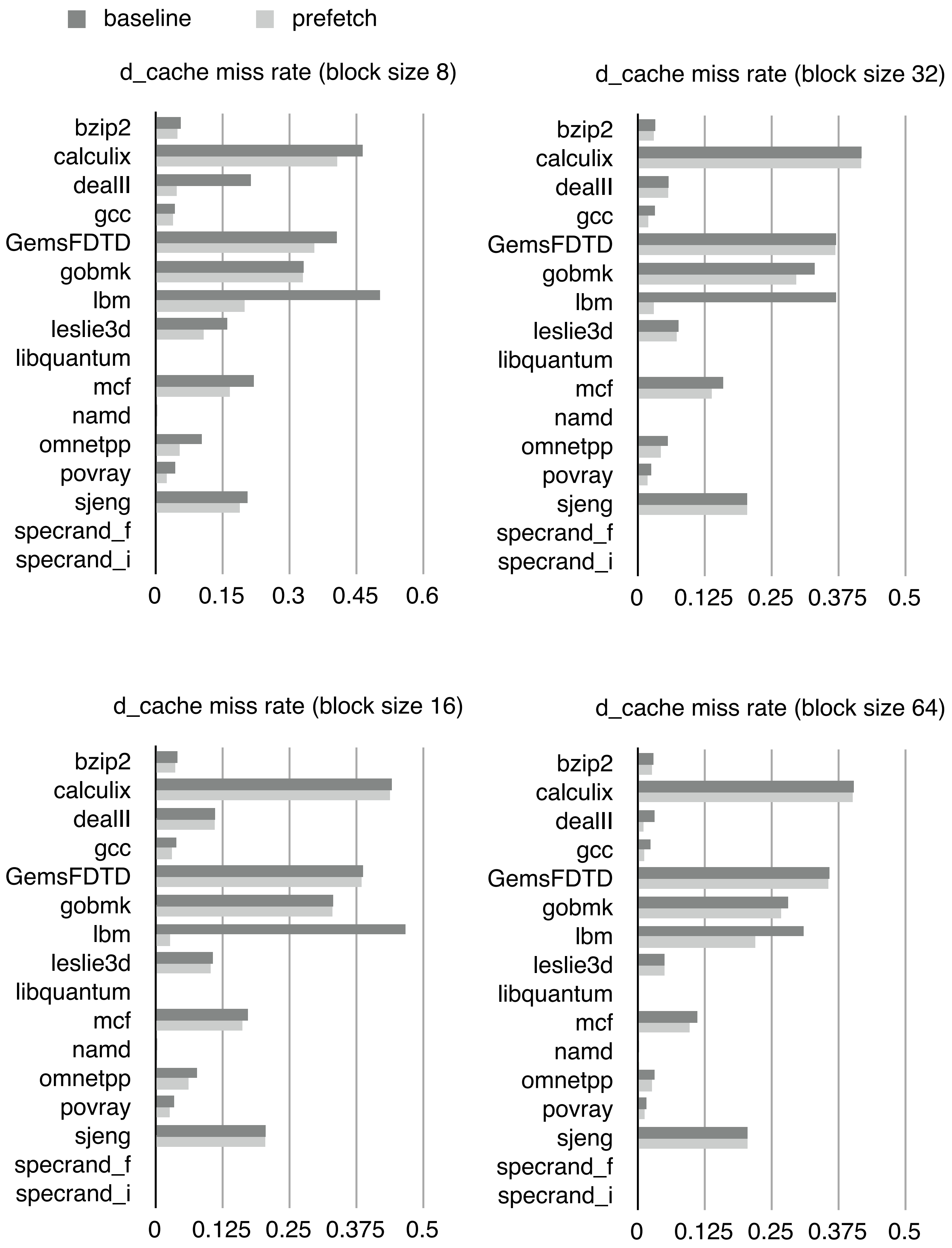
Part 1: Integer Benchmarks[1]

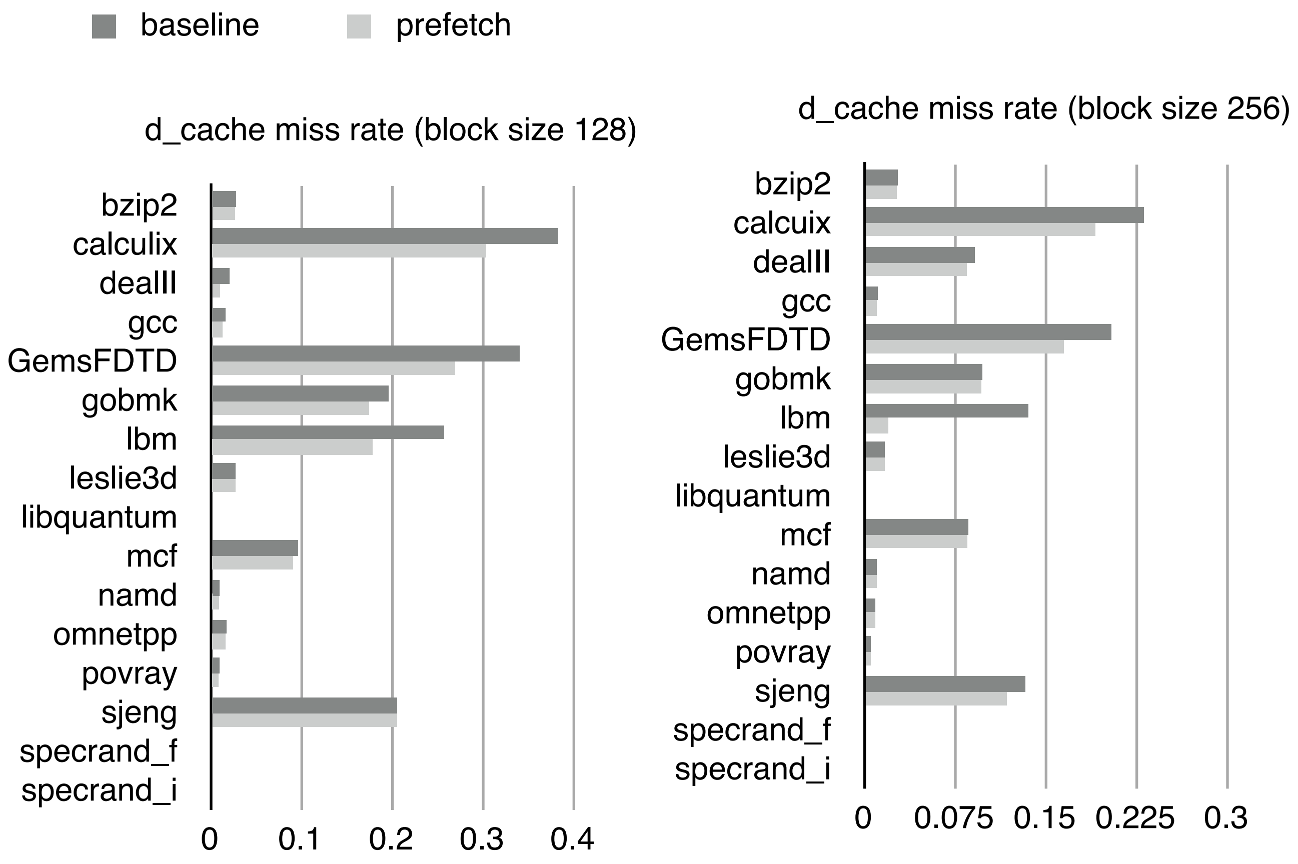
401.bzip2
403.gcc
429.mcf
445.gobmk
458.sjeng
462.libquantum
471.omnetpp

Part 2: Floating Point Benchmarks

444.namd
447.dealII
453.povray
454.calculix
459.GemsFDTD
470.lbm
999.specrand

[1]SPEC CPU2006 Benchmark Descriptions. Descriptions written by the SPEC CPU Subcommittee and by the original program authors. Edited by John L. Henning, Secretary, SPEC CPU Subcommittee, and Performance Engineer, Sun Microsystems.


■ baseline
■ prefetch
d_cache miss rate (block size 8)
d_cache miss rate (block size 32)
d_cache miss rate (block size 16)
d_cache miss rate (block size 64)
bzip2
calculix
dealII
gcc
GemsFDTD
gobmk
lbm
leslie3d
libquantum
mcf
namd
omnetpp
povray
sjeng
specrand_f
specrand_i
0
0.15
0.3
0.45
0.6
0.125
0.25
0.375
0.5



The graphs show cache miss rate for each different benchmark with different cache sizes. As you can see from the graphs, the prefetching all around improves the cache miss rate for the data cache. The difference is most visible in the lbm benchmark for smaller block sizes, whereas sjeng shows only minimal change.

The paper also goes into detail about performance boosts of installing a prefetcher versus the performance gain of doubling the cache block size. In our experiment, most of the benchmarks show minor performance gain with prefetching against larger block size, though it is not consistent, and for some benchmarks, such as bzip2, prefetching actually doesn't help performance more than increasing block size. There are patterns similar to the ones shown in the original paper, but to a much smaller extent. For the benchmarks given in the paper, there was generally around a 50-100% performance gain achievable between prefetching and increasing block size. In our results, outside of a few outliers, most performance increases were less than 10% when they were there. Going from block size 16 with prefetching to 32 without, there were generally not gains.

The paper also references prefetch overhead, that is, the blocks that are prefetched but never later referenced. These blocks take up space in the cache, kicking out other potentially useful cache blocks. In our experiment, we did not keep track of these, though we suspect from the performance gains that the prefetch overhead is generally very small.