Cache Prefetching

ECE552 Lab 4

Jay Mohile

Letian Zhang

I. BACKGROUND

In recent years, computer performance has been significantly bottlenecked by the "memory wall": memory access time has scaled significantly slower than processor clock speed. As such, even when a processor can request memory at a higher rate than ever, it is limited by memory's ability to fulfill those requests.

A common solution is multi-level caching: shuttling data into smaller, faster memory blocks for faster lookup. However, this only accelerates 2nd+ accesses, and leaves the first access of a new region of memory slow. Enter *cache-prefetching*, mechanisms to speculatively cache data before it is requested, to speed up 1st access. Three prefetch algorithms are evaluated below.

II. MICRO-BENCHMARKS

Below, simple benchmarks are discussed to evaluate correct implementation of each prefetcher. Note that these cover the general case, but are not exhaustive (i.e testing every possible edge-case of data access patterns, block overlap, state transitions, etc). Furthermore, cache-access is abstract enough that our analysis can be (mostly) constrained to the C-level. Please note that in all cases, the benchmarks were tuned to the default configurations (e.g block size, access interval), rather than the other way around.

Next-Line Prefetcher (Q1)

Whenever a data block B is accessed by software, next-line prefetchers also fetch block B+1. While simple, this can be beneficial when sequential access patterns are present (given compatible working-set size).

The default cache-configuration (cache-lru-nextline) defines a cache of 256 entries x 64 byte blocks. Our benchmark sets up a char array perfectly sized for this cache (less one block, for instructions). We iterate this array multiple times, accessing a single block on each iteration. The first time through, N prefetch misses are expected (since data must be loaded), while only one SW-level L1 miss is expected, since that miss will trigger future prefetches. On subsequent loops, no misses should occur. These expectations are verified in our benchmark.

Stride Prefetcher (Q2)

One limitation of next-line prefetchers is they ignore the possibility of non-single-block access intervals. Stride prefetchers remedy this by learning this interval, per instruction, at runtime. In our benchmark, we seek to validate two aspects of the stride prefetcher.

Firstly, the stride learned for a particular instruction can change over time. For the former, we create a large array, and access it sequentially at an interval of X bytes. Every N cycles, we change this interval. We correctly see a single L1-miss every time this interval is changed, and one additional miss when the RPT entry is first created.

Secondly, the prefetcher disables itself in the presence of an irregular pattern. Here, we create another large array, and access it by a sequentially increasing interval (i.e 1, 2, …) to ensure no cache hits. As expected we see L1-misses on every entry, but only a single prefetch-miss before it transitions to NO_PRED.

Open Ended Prefetcher (Q6)

Our open-ended prefetcher builds upon the stride mechanism, but adds multiple non-trivial improvements. While true novelty is not claimed, these changes were made independently.

1. **Probation**: The stride-prefetcher immediately evicts an RPT entry on tag miss. Our open-ended approach instead keeps a *probation* counter per entry, which is incremented on misses, and decremented on hits. Only when this exceeds a threshold is the entry considered sufficiently 'useless' to be evicted.

2. **Scheduling:** It was observed that several hundred accesses could occur between two occurrences of the same PC. Given the cache has only 256 blocks, this could result in suboptimal speculation (see section V). As such, our approach tracks the expected temporal access interval, T, of each RPT entry. If that interval exceeds some threshold, it "schedules" the prefetch forward by ~T memory accesses.

This was validated by accessing a large data-array at a fixed interval, but occasionally adding a noisy access, to simulate an unhelpful RPT alias. This noise was correctly ignored. Furthermore, this benchmark was manually stepped to validate scheduling behaviour.

These factors together appeared to improve cache access time by 10% relative to the stride prefetcher. Adding probation consisted of an additional integer of space per RPT entry (although, this could be reduced by using less bits for this counter). Adding scheduling took more significant space, a "last access time" field per RPT entry, and an array to schedule future prefetches. These were optimized for simplicity, and offer ample room for improvements such as:

1. A truncated last-access-time.
2. A schedule that offers a limited number of schedule slots, rather than indexing by time.
3. An optimized schedule that stores references to the RPT, rather than prefetch addresses.
4. Tracking redundantly scheduled accesses, and eliminating them.

III. RESULTS (Q3)

Applying access times of L1=1, L2=10, and Memory=100, the following cache access times are measured against the *compress* benchmark. Average access times were computed as follows.

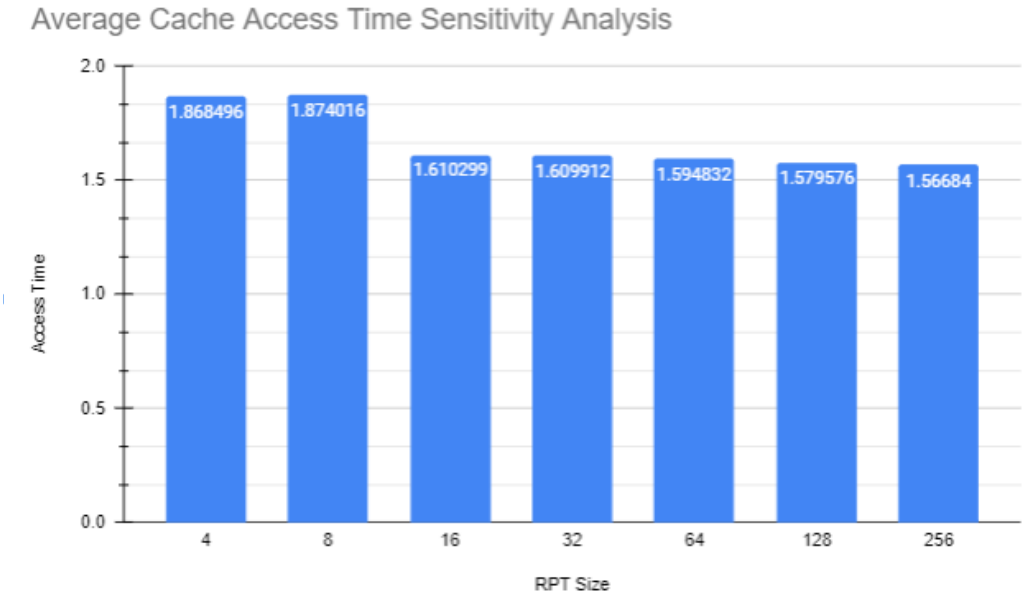
$$E[T_{L2}] = T_{L2,\,Hit} + r_{L2,\,Miss} T_{Mem}$$

$$E[T_{Access}] = T[T_{L1}] = T_{L1,\,Hit} + r_{L1,\,Miss} T_{L2}$$

Config	L1 Miss Rate	L2 Miss Rate	Average Access Time
Baseline	4.16%	11.40%	1.89024
Next Line	4.16%	8.41%	1.765856
Stride	3.87%	5.77%	1.610299

IV. SENSITIVITY ANALYSIS (Q4)

The stride-prefetcher uses a table (RPT) of some fixed size to track metadata about address access patterns. The relationship between this table size and the overall prefetch performance is of interest.



From this graph, we can see the average access time improved until an RPT size of 16, but plateaus beyond that. There are several possible explanations for this, including:

1. **Memory Coverage:** Only ~16 blocks of memory are accessed by the benchmark, given the limited instruction count.
2. **Working Set Coverage:** The working set of instructions needing memory access is ~16.
3. **Stride Coverage:** Given an RPT size of 16, we cover all access intervals (with other instructions 'piggybacking').

V. DISCUSSION (Q5)

A key aspect of prefetchers is that they are speculative. They fetch data that has not been explicitly requested, in the hope that it will soon be required. However, in doing this, they may end up evicting data that is soon to be explicitly required. While this can be partially inferred through observing miss rates, hit rates, replacement rates, etc, it would be helpful to have more concrete statistics tracking this behaviour. For example, the following could be useful.

1. **Failed/premature speculations:** prefetches that were evicted before being used.
2. **Greedy speculations:** prefetches that evicted an item that was actually needed before the prefetch.

APPENDIX A: RAW RESULTS

Baseline

```
sim: ** simulation statistics **
sim_num_insn              80432288 # total number of instructions executed
sim_num_refs              28767230 # total number of loads and stores executed
sim_elapsed_time                 2 # total simulation time in seconds
sim_inst_rate         40216144.0000 # simulation speed (in insts/sec)
il1.accesses              80432288 # total number of accesses
il1.hits                  80429660 # total number of hits
il1.misses                    2628 # total number of misses
il1.replacements              2372 # total number of replacements
il1.writebacks                   0 # total number of writebacks
il1.invalidations                0 # total number of invalidations
il1.miss_rate               0.0000 # miss rate (i.e., misses/ref)
il1.repl_rate               0.0000 # replacement rate (i.e., repls/ref)
il1.wb_rate                 0.0000 # writeback rate (i.e., wrbks/ref)
il1.inv_rate                0.0000 # invalidation rate (i.e., invs/ref)
il1.read_accesses         80432288 # total number of read accesses
il1.read_hits             80429660 # total number of read hits
il1.read_misses               2628 # total number of read misses
il1.read_miss_rate          0.0000 # read miss rate
il1.prefetch_accesses            0 # total number of prefetch accesses
il1.prefetch_hits                0 # total number of prefetch hits
il1.prefetch_misses              0 # total number of prefetch misses
dl1.accesses              29063125 # total number of accesses
dl1.hits                  27853167 # total number of hits
dl1.misses                 1209958 # total number of misses
dl1.replacements           1209702 # total number of replacements
dl1.writebacks              592063 # total number of writebacks
dl1.invalidations                0 # total number of invalidations
dl1.miss_rate               0.0416 # miss rate (i.e., misses/ref)
dl1.repl_rate               0.0416 # replacement rate (i.e., repls/ref)
dl1.wb_rate                 0.0204 # writeback rate (i.e., wrbks/ref)
dl1.inv_rate                0.0000 # invalidation rate (i.e., invs/ref)
dl1.read_accesses          17381698 # total number of read accesses
```

```
dl1.read_hits            16513846 # total number of read hits
dl1.read_misses            867852 # total number of read misses
dl1.read_miss_rate          0.0499 # read miss rate
dl1.prefetch_accesses            0 # total number of prefetch accesses
dl1.prefetch_hits                0 # total number of prefetch hits
dl1.prefetch_misses              0 # total number of prefetch misses
ul2.accesses               1804649 # total number of accesses
ul2.hits                   1598867 # total number of hits
ul2.misses                  205782 # total number of misses
ul2.replacements            201686 # total number of replacements
ul2.writebacks              184181 # total number of writebacks
ul2.invalidations                0 # total number of invalidations
ul2.miss_rate               0.1140 # miss rate (i.e., misses/ref)
ul2.repl_rate               0.1118 # replacement rate (i.e., repls/ref)
ul2.wb_rate                 0.1021 # writeback rate (i.e., wrbks/ref)
ul2.inv_rate                0.0000 # invalidation rate (i.e., invs/ref)
ul2.read_accesses           1212586 # total number of read accesses
ul2.read_hits               1006950 # total number of read hits
ul2.read_misses              205636 # total number of read misses
ul2.read_miss_rate          0.1696 # read miss rate
ul2.prefetch_accesses            0 # total number of prefetch accesses
ul2.prefetch_hits                0 # total number of prefetch hits
ul2.prefetch_misses              0 # total number of prefetch misses
ld_text_base            0x00400000 # program text (code) segment base
ld_text_size               103840 # program text (code) size in bytes
ld_data_base            0x10000000 # program initialized data segment base
ld_data_size             44123012 # program init'ed `.data' and uninit'ed `.bss' size in bytes
ld_stack_base           0x7fffc000 # program stack segment base (highest address in stack)
ld_stack_size              16384 # program initial stack size
ld_prog_entry           0x00400140 # program entry point (initial PC)
ld_environ_base         0x7fff8000 # program environment base address address
ld_target_big_endian             0 # target executable endian-ness, non-zero if big endian
mem.page_count                 206 # total number of pages allocated
mem.page_mem                   824k # total size of memory pages allocated
mem.ptab_misses                206 # total first level page table misses
```

mem.ptab_accesses 380232234 # total page table accesses
mem.ptab_miss_rate 0.0000 # first level page table miss rate

Next Line

sim: ** simulation statistics **
sim_num_insn 80432288 # total number of instructions executed
sim_num_refs 28767230 # total number of loads and stores executed
sim_elapsed_time 3 # total simulation time in seconds
sim_inst_rate 26810762.6667 # simulation speed (in insts/sec)
il1.accesses 80432288 # total number of accesses
il1.hits 80429660 # total number of hits
il1.misses 2628 # total number of misses
il1.replacements 2372 # total number of replacements
il1.writebacks 0 # total number of writebacks
il1.invalidations 0 # total number of invalidations
il1.miss_rate 0.0000 # miss rate (i.e., misses/ref)
il1.repl_rate 0.0000 # replacement rate (i.e., repls/ref)
il1.wb_rate 0.0000 # writeback rate (i.e., wrbks/ref)
il1.inv_rate 0.0000 # invalidation rate (i.e., invs/ref)
il1.read_accesses 80432288 # total number of read accesses
il1.read_hits 80429660 # total number of read hits
il1.read_misses 2628 # total number of read misses
il1.read_miss_rate 0.0000 # read miss rate
il1.prefetch_accesses 0 # total number of prefetch accesses
il1.prefetch_hits 0 # total number of prefetch hits
il1.prefetch_misses 0 # total number of prefetch misses
dl1.accesses 29063125 # total number of accesses
dl1.hits 27853808 # total number of hits
dl1.misses 1209317 # total number of misses
dl1.replacements 2343688 # total number of replacements
dl1.writebacks 610683 # total number of writebacks
dl1.invalidations 0 # total number of invalidations
dl1.miss_rate 0.0416 # miss rate (i.e., misses/ref)
dl1.repl_rate 0.0806 # replacement rate (i.e., repls/ref)
dl1.wb_rate 0.0210 # writeback rate (i.e., wrbks/ref)

dl1.inv_rate 0.0000 # invalidation rate (i.e., invs/ref)
dl1.read_accesses 17381698 # total number of read accesses
dl1.read_hits 16486343 # total number of read hits
dl1.read_misses 895355 # total number of read misses
dl1.read_miss_rate 0.0515 # read miss rate
dl1.prefetch_accesses 29063125 # total number of prefetch accesses
dl1.prefetch_hits 27928498 # total number of prefetch hits
dl1.prefetch_misses 1134627 # total number of prefetch misses
ul2.accesses 1822628 # total number of accesses
ul2.hits 1669292 # total number of hits
ul2.misses 153336 # total number of misses
ul2.replacements 245356 # total number of replacements
ul2.writebacks 197615 # total number of writebacks
ul2.invalidations 0 # total number of invalidations
ul2.miss_rate 0.0841 # miss rate (i.e., misses/ref)
ul2.repl_rate 0.1346 # replacement rate (i.e., repls/ref)
ul2.wb_rate 0.1084 # writeback rate (i.e., wrbks/ref)
ul2.inv_rate 0.0000 # invalidation rate (i.e., invs/ref)
ul2.read_accesses 1211945 # total number of read accesses
ul2.read_hits 1058778 # total number of read hits
ul2.read_misses 153167 # total number of read misses
ul2.read_miss_rate 0.1264 # read miss rate
ul2.prefetch_accesses 1134627 # total number of prefetch accesses
ul2.prefetch_hits 1038511 # total number of prefetch hits
ul2.prefetch_misses 96116 # total number of prefetch misses
ld_text_base 0x00400000 # program text (code) segment base
ld_text_size 103840 # program text (code) size in bytes
ld_data_base 0x10000000 # program initialized data segment base
ld_data_size 44123012 # program init'ed `.data' and uninit'ed `.bss' size in bytes
ld_stack_base 0x7fffc000 # program stack segment base (highest address in stack)
ld_stack_size 16384 # program initial stack size
ld_prog_entry 0x00400140 # program entry point (initial PC)
ld_environ_base 0x7fff8000 # program environment base address address
ld_target_big_endian 0 # target executable endian-ness, non-zero if big endian
mem.page_count 206 # total number of pages allocated

mem.page_mem 824k # total size of memory pages allocated
mem.ptab_misses 206 # total first level page table misses
mem.ptab_accesses 380232234 # total page table accesses
mem.ptab_miss_rate 0.0000 # first level page table miss rate

Stride

sim: ** simulation statistics **
sim_num_insn 80432288 # total number of instructions executed
sim_num_refs 28767230 # total number of loads and stores executed
sim_elapsed_time 3 # total simulation time in seconds
sim_inst_rate 26810762.6667 # simulation speed (in insts/sec)
il1.accesses 80432288 # total number of accesses
il1.hits 80429660 # total number of hits
il1.misses 2628 # total number of misses
il1.replacements 2372 # total number of replacements
il1.writebacks 0 # total number of writebacks
il1.invalidations 0 # total number of invalidations
il1.miss_rate 0.0000 # miss rate (i.e., misses/ref)
il1.repl_rate 0.0000 # replacement rate (i.e., repls/ref)
il1.wb_rate 0.0000 # writeback rate (i.e., wrbks/ref)
il1.inv_rate 0.0000 # invalidation rate (i.e., invs/ref)
il1.read_accesses 80432288 # total number of read accesses
il1.read_hits 80429660 # total number of read hits
il1.read_misses 2628 # total number of read misses
il1.read_miss_rate 0.0000 # read miss rate
il1.prefetch_accesses 0 # total number of prefetch accesses
il1.prefetch_hits 0 # total number of prefetch hits
il1.prefetch_misses 0 # total number of prefetch misses
dl1.accesses 29063125 # total number of accesses
dl1.hits 27939674 # total number of hits
dl1.misses 1123451 # total number of misses
dl1.replacements 1442871 # total number of replacements
dl1.writebacks 595757 # total number of writebacks
dl1.invalidations 0 # total number of invalidations
dl1.miss_rate 0.0387 # miss rate (i.e., misses/ref)

```
dl1.repl_rate              0.0496 # replacement rate (i.e., repls/ref)
dl1.wb_rate                0.0205 # writeback rate (i.e., wrbks/ref)
dl1.inv_rate               0.0000 # invalidation rate (i.e., invs/ref)
dl1.read_accesses         17381698 # total number of read accesses
dl1.read_hits             16476513 # total number of read hits
dl1.read_misses             905185 # total number of read misses
dl1.read_miss_rate          0.0521 # read miss rate
dl1.prefetch_accesses      10790157 # total number of prefetch accesses
dl1.prefetch_hits          10470481 # total number of prefetch hits
dl1.prefetch_misses          319676 # total number of prefetch misses
ul2.accesses               1721836 # total number of accesses
ul2.hits                   1622475 # total number of hits
ul2.misses                   99361 # total number of misses
ul2.replacements            247941 # total number of replacements
ul2.writebacks              202167 # total number of writebacks
ul2.invalidations                0 # total number of invalidations
ul2.miss_rate               0.0577 # miss rate (i.e., misses/ref)
ul2.repl_rate               0.1440 # replacement rate (i.e., repls/ref)
ul2.wb_rate                 0.1174 # writeback rate (i.e., wrbks/ref)
ul2.inv_rate                0.0000 # invalidation rate (i.e., invs/ref)
ul2.read_accesses           1126079 # total number of read accesses
ul2.read_hits               1026864 # total number of read hits
ul2.read_misses               99215 # total number of read misses
ul2.read_miss_rate          0.0881 # read miss rate
ul2.prefetch_accesses        319676 # total number of prefetch accesses
ul2.prefetch_hits            167000 # total number of prefetch hits
ul2.prefetch_misses          152676 # total number of prefetch misses
ld_text_base           0x00400000 # program text (code) segment base
ld_text_size               103840 # program text (code) size in bytes
ld_data_base           0x10000000 # program initialized data segment base
ld_data_size             44123012 # program init'ed `.data' and uninit'ed `.bss' size in bytes
ld_stack_base          0x7fffc000 # program stack segment base (highest address in stack)
ld_stack_size              16384 # program initial stack size
ld_prog_entry          0x00400140 # program entry point (initial PC)
ld_environ_base         0x7fff8000 # program environment base address address
```

```
ld_target_big_endian            0 # target executable endian-ness, non-zero if big endian
mem.page_count                206 # total number of pages allocated
mem.page_mem                  824k # total size of memory pages allocated
mem.ptab_misses               206 # total first level page table misses
mem.ptab_accesses         380232234 # total page table accesses
mem.ptab_miss_rate           0.0000 # first level page table miss rate
```