ENGINEERING

McMaster University

Project: Prefetchers

COMPENG 4DM4 – Computer Architecture

Matteo Toffolon (toffolom, 400303361)

Jason Liu (liuj321, 400254903)

Jaskaran Dhillon (dhillj19, 400189244)

Olukemi Odujinrin (odujinro, 400317700)

December 5, 2023

Declaration of Contribution Statement

Group Member	***Contribution***
Jason Liu	Wrote script to analyze trace files and identify optimal parameters. Tested parameters to find AMAT times.
Jaskaran Dhillon	Worked to improve code efficiency. Helped run the benchmark tests to find optimal parameters.
Matteo Toffolon	Implemented current baseline stride-stream buffer prefetching algorithm (Matteo-Dev branch)
Olukemi Odujinrin	Implemented a draft stride prefetching algorithm (see ***Kemi-Dev*** branch)

Description of Prefetcher

The prefetcher algorithm is a combination of stride prefetching and stream buffer prefetching. The stride prefetching is based on a reference prediction table and the use of the program counter. This implementation of our prefetcher is supported and inspired by an article from IEEE explore, *Prefetching using Markov Predictors*, specifically sections 3.2 and 3.3. Stride prefetching alone tends to utilize more resources than the stream buffering technique. Stream buffers are designed to prefetch sequential streams of cache lines, independent of program context [1]. As a result, we choose to implement a stride prefetcher in series with stream buffers to yield better results and cache access performance.

Changes to improve performance include removing the ***state*** attribute of the reference prediction table struc and adding an LRU counter instead. To simplify code efficiency, the check for max LRU is executed within the for loop, instead of outside. Lastly, a variable int *hit_count* is used to reduce starvation for the tiny trace file. The block of code used to compare *hit_count* and a certain threshold helps improve the result for the tiny trace file. A better implementation can be executed, as of now this a quick solution to this problem.

State Accounting

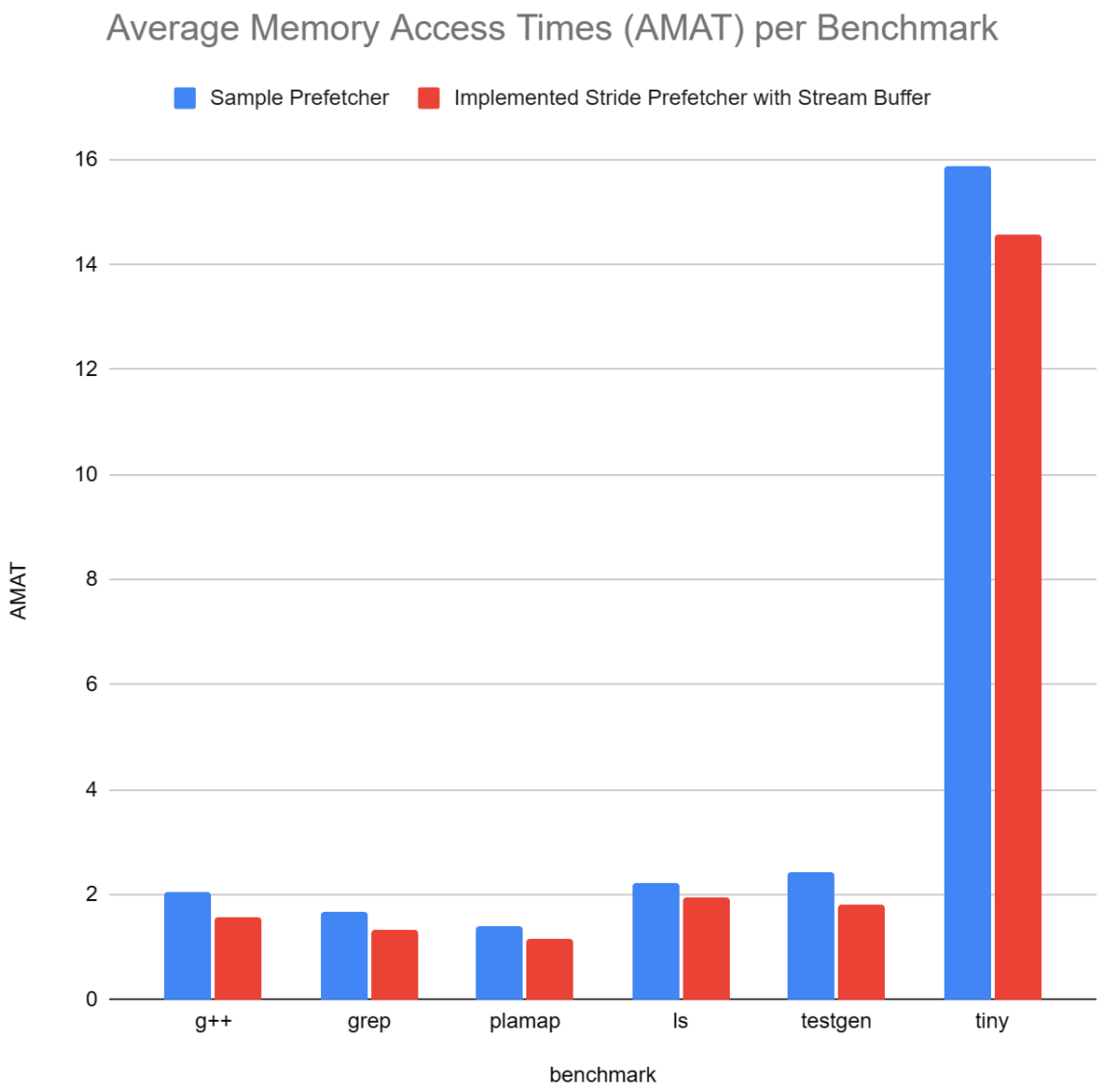
Reference Prediction Table

The reference prediction table is used to keep a history of recently executed instructions with each entry holding four 32-bit integers. An instruction's program counter, memory reference address, stride and LRU count make up an entry in the table to yield 16B per entry x 128 entries for a total of 2KB of saved memory. When a L1 miss occurs, the instruction's program counter is checked with every table entry's program counter and if a match is found, a prefetch for the effective address + the table stride is launched. The table uses LRU replacement policy when the table becomes full, with the LRU counts of each entry compared to one other to determine the correct spot for replacement.

Prefetch Buffer

The prefetch buffer is a small buffer used to store subsequent prefetch addresses. It consists of 10 entries of 32-bit addresses (4B) for a total of 40B of additional prefetch storage. It was experimentally determined that for most traces, a better speedup can be achieved if subsequent L1 cache line addresses are prefetched in between CPU requests regardless of if it is an L1 hit or miss, hence the use of the prefetch buffer.

AMAT Graph



Commands

./cacheSim traces/g++.trace &>out/g++.trace.out

./cacheSim traces/grep.trace &>out/grep.trace.out

./cacheSim traces/ls.trace &>out/ls.trace.out

./cacheSim traces/plamap.trace &>out/plamap.trace.out

./cacheSim traces/testgen.trace &>out/testgen.trace.out

./cacheSim traces/tiny.trace &>out/tiny.trace.out

Trace Analysis Script

Note: Trace analysis Python script may also be found in the project repository.

```
from ast import literal_eval
ran = False
prev = 0
current = 0
large = 0
max_stride = 64
counter = [0] * (max_stride + 1)

with open("g++.trace") as infile: #Modified to determine trace file anaylzed
    for line in infile:
        try:
            current = int(line[2:11],0) #Varied ending from 11 to 12 as needed
        except ValueError:
            current = current
        stride = current - prev #Check concurrent stride values
        print("Stride = " + str(stride)) #For debugging
        prev = current
        if(stride > 64):
            large += 1 #For values larger than 64, note as "large"
        else:
            for stride_value in range(max_stride + 1): #Comparison up to 64
                if stride == stride_value or (-1 * stride) == stride_value:
                    counter[stride_value] = counter[stride_value] + 1

print(counter)
print(large)
```

References

[1] D. J. Joseph and D. Grunwald, “Prefetching using Markov predictors,” *IEEE Transactions on Computers*, vol. 48, no. 2, pp. 121–133, Jan. 1999, doi: https://doi.org/10.1109/12.752653.