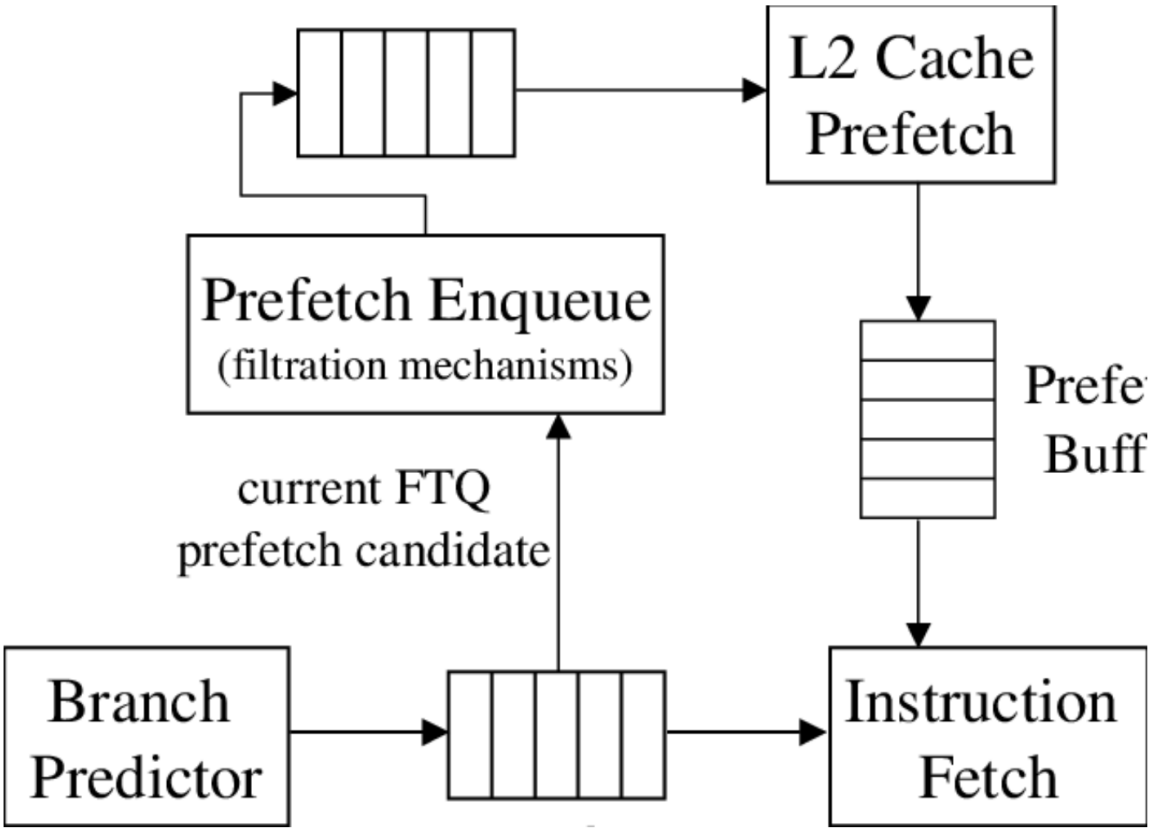
CS 203 - Advanced Computer Architecture
Prefetcher Implementation



S HANISHA SREE
SID - 862191473

I have taken Prefetching Using Markov Predictors IEEE paper as a reference and various other websites in google search engine.

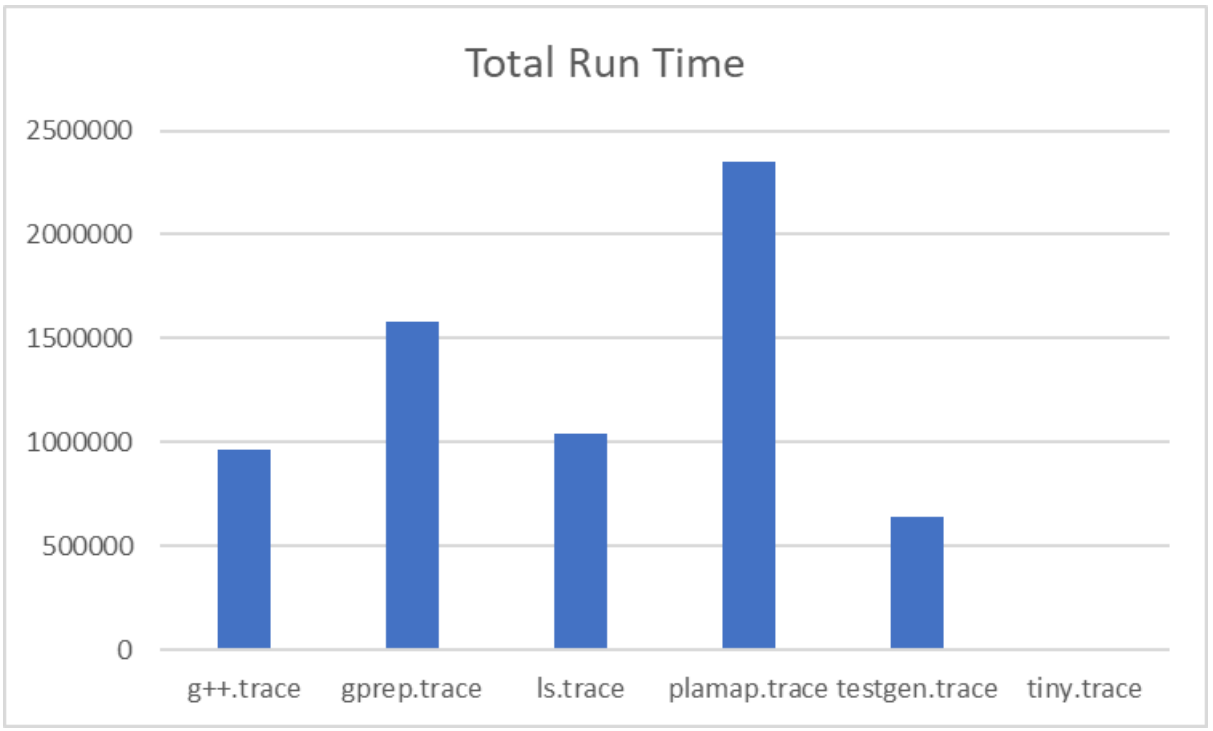
There are many techniques that have been proposed to decrease memory latency, such as caches, locality optimizations, pipelining, out-of-order execution, and multithreading.
Prefetching is one approach to reducing the latency of memory operations in modern computer systems. Prefetching brings data or instructions closer to the processor before it is needed so that the processor will not have to stall and wait for the data, thereby reducing the cache miss rate and decreasing memory access latency.

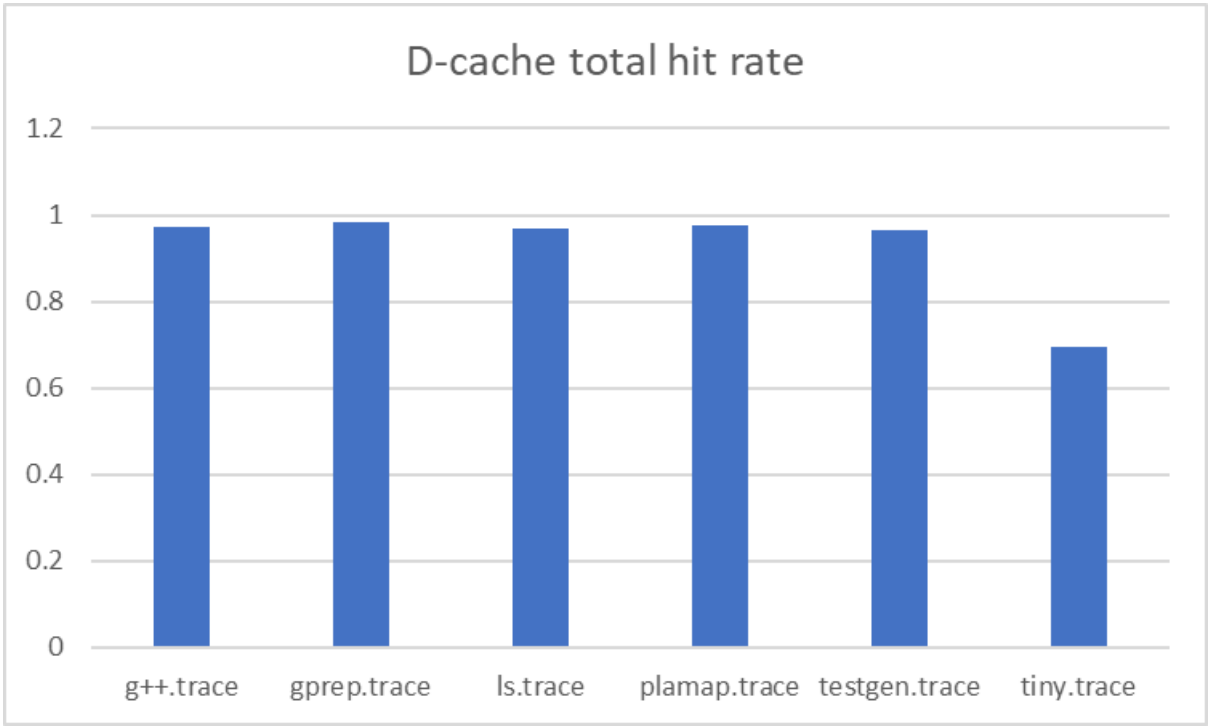
There are two main types of prefetching: software prefetching and hardware prefetching. And there are two types of hardware prefetching: data prefetching and instruction prefetching. I have implemented one data prefetching algorithm based on the Differential Markov Model.

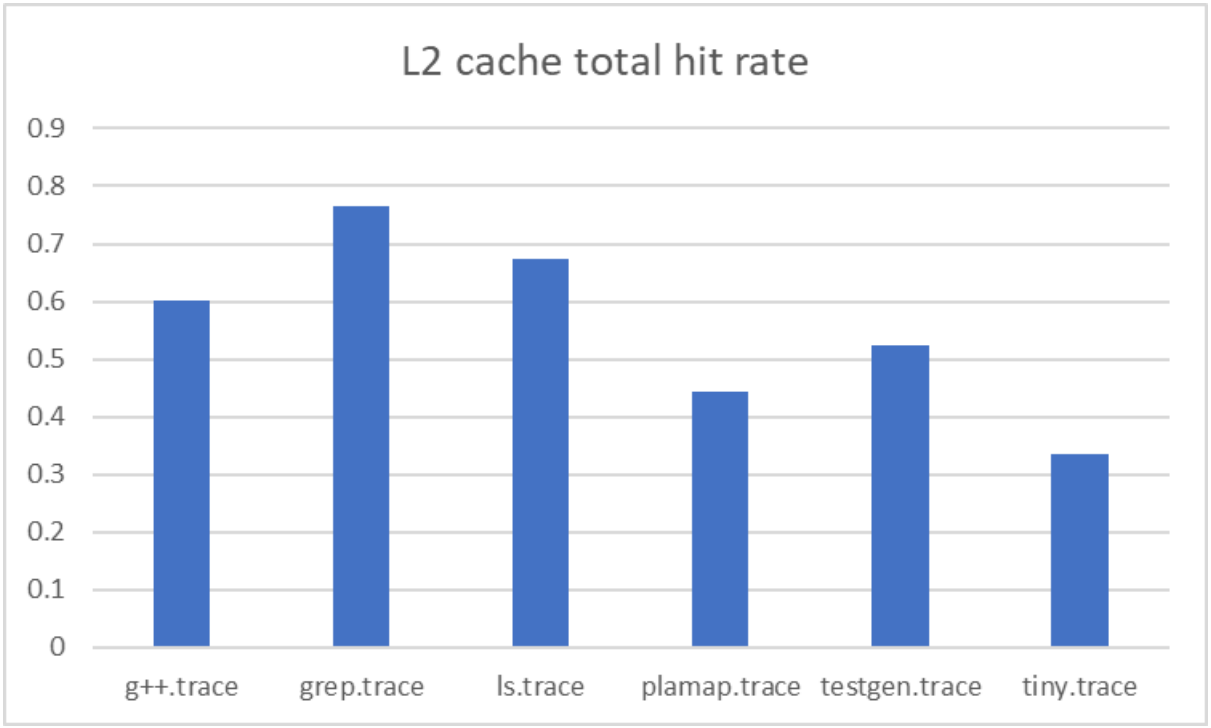
A significant advantage of hardware prefetching techniques over software techniques is that they do not need support from the programmer or compiler.
This prefetcher acts as an interface between the on-chip and off-chip cache and can be added to existing computer designs. The Markov prefetcher is distinguished by prefetching multiple reference predictions from the memory subsystem, and then prioritizing the delivery of those references to the processor. This design results in a prefetching system that provides good coverage, is accurate, and produces timely results that can be effectively used by the processor.
It is simple, effective, and realizable Markov prefetcher can be built as an off-chip component of the memory subsystem.

The following are the graphical representations of simulating statistics.

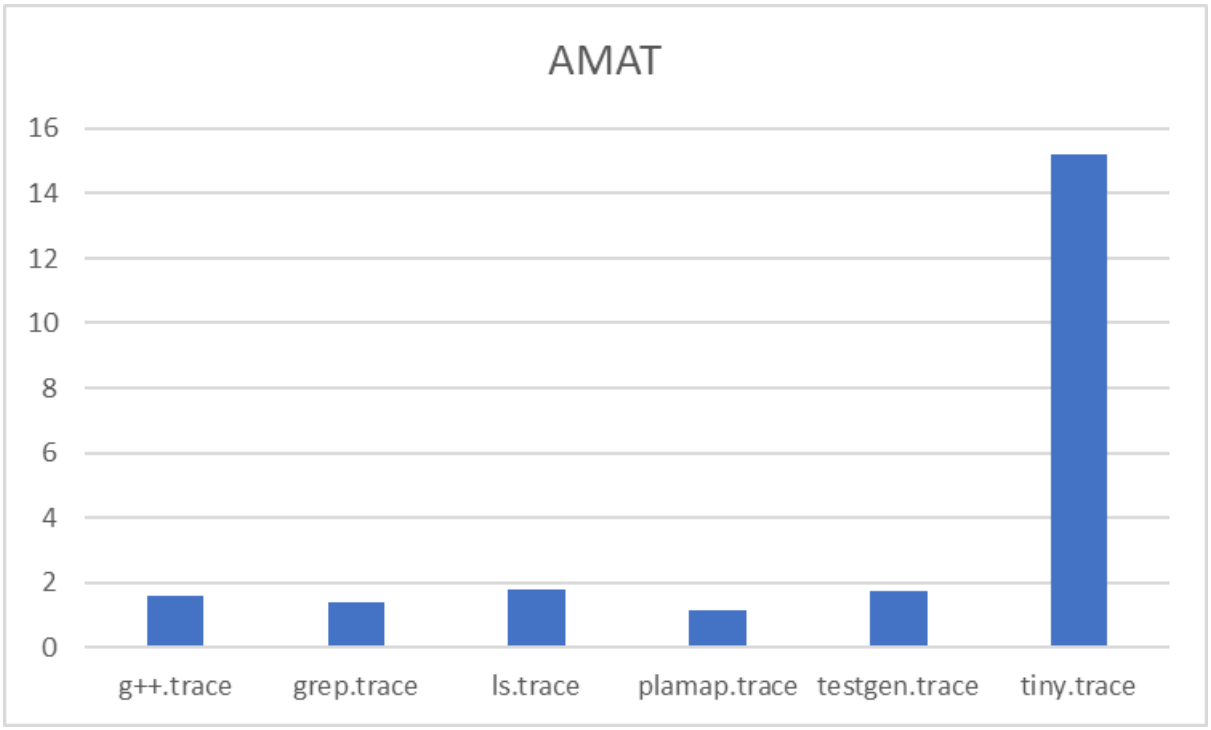
Total Run Time: The total run time of the program, assuming that the last memory access was the last instruction of the program. The unit for this should be cycles.

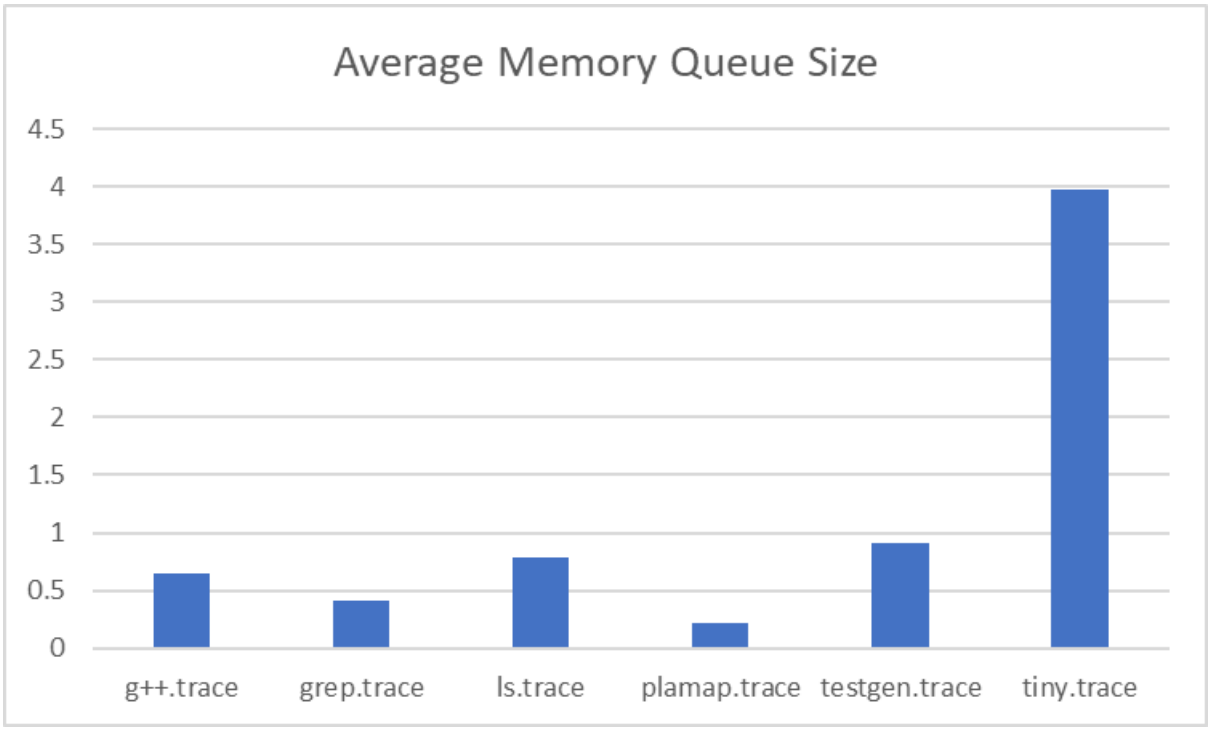



D-cache total hit rate
1.2
1
0.8
0.6
0.4
0.2
0
g++.trace
gprep.trace
ls.trace
plamap.trace
testgen.trace
tiny.trace


L2 cache total hit rate
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
g++.trace
grep.trace
ls.trace
plamap.trace
testgen.trace
tiny.trace

Average Memory Access Latency: The average number of cycles needed to complete a memory access. This number should be to 4 decimal places of precision.





To build the project : make
To check the outputs and statistics on the traces : ./cacheSim traces/[trace file name]
Challenges : It was hard to read the paper and understand how to implement it.