Keerthinivash Korisal
862034133
CS203 - Prefetching Project Report

Prefetcher Description:

The prefetcher implemented is derived from the article "Effective Hardware-Based Prefetching for High-Performance Processors" by Tien-Fu Chen and Jean-Loup Bare. The basic idea of my prefetcher is stride prefetching. The concept is to record the distance between memory addresses referenced by a load instruction and the last address referenced by the load instructions. In doing so, the next time the same load instruction is performed, the prefetcher is able to prefetch the last address and the stride (distance) recorded earlier. These values are stored in a RPT or Reference Prefetching Table. This table contains information about recently executed load/ store instructions in order to predict the access pattern. The RPT is generally categorized as the following; tag, previous address, stride and state. Tag in an RPT is similar to usually cache organization, usually higher order PC bits. Previous address is the address of previous execution of a load/ store. Stride is the difference between previous address and one previous than that. Finally, state, usually 2 bits, which can be initial, stady, transient and no-prediction. In general, if a load/ store instruction is present in the RPT and a steady state is present, then prefetch of new address plus the stride is executed.

In my execution of an RPT based prefetcher, I did not implement a state based RPT prefetching due to the complexity. Instead, my prefetcher automatically fetches the next address plus L2 block (32 bytes), when anything other than a steady state is achieved. If the request has an L1 hit, then the prefetcher checks if the PC of the request is present in the RPT. if present, the next address request is fetched using (req_addr + stride). If the PC is not present in the table, then (req.addr + L2block) is fetched. In the scenario when the L1 miss is present, the prefetcher again checks if PC is present in the RPT, if present, it check if the stride of the specific entry in the RPT is the same by ((req.addr - prev.addr) == rpt.stride). If all else fails, the prefetcher automatically fetches address plus L2 block.

There are 128 entries in the RPT table, and the prefetcher only accounts for 2 requests per L1 miss.

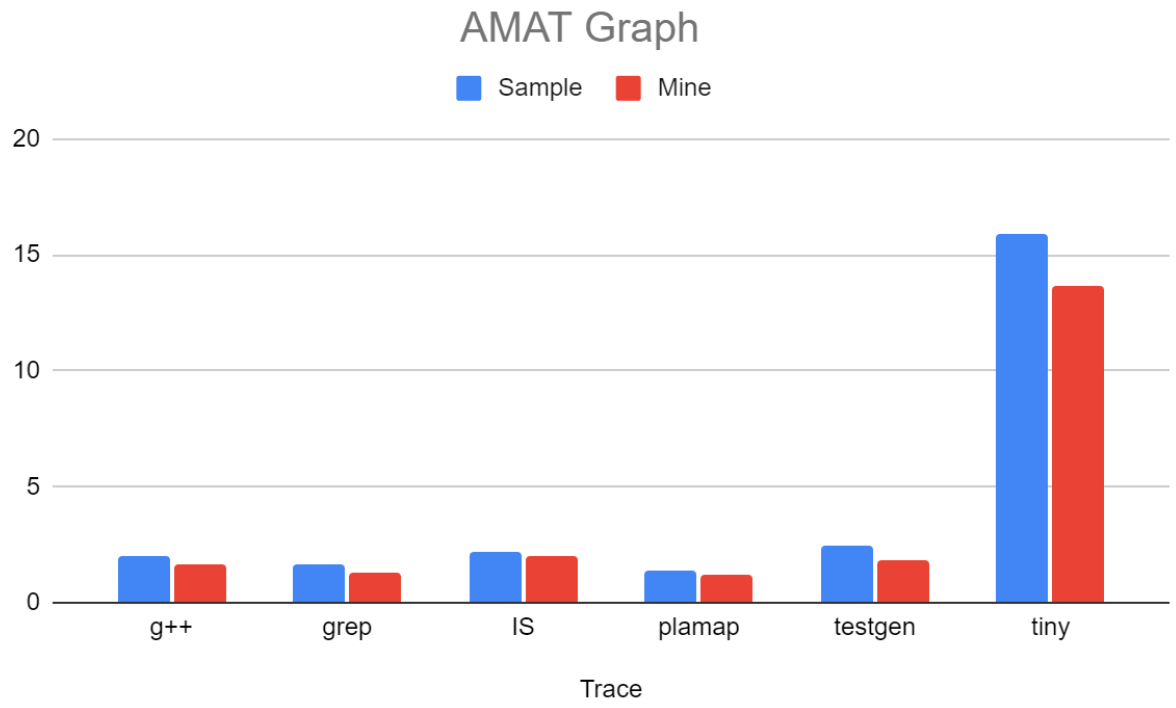
State Accounting:

RPT table entries of (pc, prev_addr, and stride) make up of three 4 byte integers. Hence, 12 bytes per entry. With the total number of entries being 128, the total state contribution is 1536 bytes.

AMAT:

	Sample/Baseline	Mine	Speedup	%Speedup
g++	2.050878	1.621279	1.264975368	26.49753682
grep	1.67389	1.334492	1.254327489	25.43274894
IS	2.215026	1.977229	1.120267809	12.02678091
plamap	1.396336	1.195728	1.167770597	16.77705967

testgen	2.430332	1.79846	1.351340591	35.13405914
tiny	15.878	13.624	1.165443335	16.54433353



It can be derived from the results, that my prefetcher had an average speedup of 22%, maximum speed up of 35% for testgen.trace and minimum speedup of 12% for ls.trace.