Design Description:

This is an algorithm called Tagged cache stride, which combined with tagged sequential prefetching with cache stride prefetching. It gives the biggest advantages of those two algorithm.
Tagged sequential prefetching is an algorithm to prefetch when the address is accessed for the first time.
Cache prefetching is an algorithm which uses the stride between the addresses in the cache to predict the prefetch stride.
This algorithm work good for normal program,especially which contains a big loop. But the disadvantage is abvious, because there are many stride patterns of the address in the cache, and this algorithm only takes two pattern into consideration which are equidistant and equidistant alternately. However, since it is common for loop to use equidistant or alternately equidistant to load address, this algorithm makes sense.
The algorithm has several steps:
(1)Find the address in the cache tracker, if there is a such address, set the tag to 0.
(2)Find the address in the cache tracker, if there is no such address, add it to cache tracker and set the tag to 1.
(3)If the tag is 0, no prefetching.
(4)If the tag is 1, record the stride between the new address with the MRU address as stride(new).

a. Compare stride(new) with stride(MRU)
   1.If stride(new)=stride(MRU): set the confidence of the new stride,
   confidence(new)=confidence(MRU)+1
   2.If stride(new)!=stride(MRU): compare the stride(new) with stride(previous MRU),
   if stride(new)=stride(previous MRU): set confidence(new)=2

b. If confidence(new)=2:
   1. prefetch with the new stride for two times.
   2. for stride(new)= stride(MRU): set confidence(new)=confidence(new)-1
   for stride(new)=stride(previous MRU): set confidence(new)=0

c. Otherwise, set stride=1.

d. Prefetch for two times.

Hardware Budget:

This algorithm uses 5 states, they are tag, address, LRU, stride, confidence.
Tag is 1 byte, address is 4 bytes(2GB memory), LRU is 4 bytes, stride is 2 bytes, confidence is 1 byte. So there are 12 bytes to form a tracker line and there are 1KB trackers so the total used hardware budget is 12KB which is 3*2^12 bits.

Result Table:

Table 1

DEFAULT	gcc	GemsFDTD	lbm	leslie3d	libquantum	mcf	milc	omnetpp	IPC
Next_line	0.22	3.36	1.66	1.36	3.30	0.28	1.16	0.77	1.51
Ip_stride	0.30	3.44	1.52	1.40	3.24	0.35	1.14	2.22	1.66
Ampm_lite	0.34	3.45	2.03	1.62	3.27	0.37	1.18	2.27	1.82
Stream	0.28	3.44	1.26	1.03	3.17	0.34	1.06	2.21	1.60
Tagged_seq	0.25	3.45	1.67	1.43	3.28	0.31	1.18	1.13	1.58
Tagged_cache	0.29	3.44	2.00	1.52	3.27	0.35	1.20	1.88	1.74

Table 2

Tagged_str_N	gcc	GemsFDTD	lbm	leslie3d	libquantum	mcf	milc	omnetpp	IPC
small_llc	0.29	3.36	2.00	1.48	3.30	0.22	1.11	1.32	1.64
low_band	0.15	3.33	0.99	0.72	3.20	0.19	0.61	1.16	1.30
scramble_load	0.29	3.34	2.00	1.46	3.30	0.35	1.18	1.86	1.72

Result Analysis:

Table 1 is the result of several algorithms run on the tracers. The best result comes from ampm_lite_prefetcher and the second result comes from tagged cache stride. The tracer with highest IPC is GemsFDTD, the results of GemsFDTD in all the algorithm are similar, which means the addresses in this tracer may be well organized in groups and the stride between every address in the group is similar and small.
The lowest column is gcc, the address in this tracer may randomly distributed and the stride between the address may be very big.
From the table we see that next line has the lowest IPC and Ampm_lite has the highest IPC. Because next-line prefetch blindly but Ampm_lite can prefetch more precisely. Ampm detects the frequency of the recent-used zone to decide the zone to prefetch, besides it uses prefetch candidates to decide the stride of prefetching. Prefetching candidates ensures the accuracy of prefetching and focusing on hot zone ensures both the prefetching accuracy and a saving on hardware budget that is why Ampm_lite has a good performance.
The tagged_cache_stride performs well, but it only takes the second place due to the limited stride pattern detection, however tagged_cache_stride shares a similarity with SPAD. SPAD has a more sophisticated process to determine the time and stride to prefetch, so it may be possible to improve tagged_cache_stride, however due to the hardware limit, a more sophisticated process may not give much improvement on performance.
Ip_stride prefetcher is a good algorithm which shows a concise and clear process to determine the prefetch stride. However Ip_stride prefetcher needs a big space to record the states (instruction pointer, address,last stride and LRU) ,which limits the number of IP that can be recorded, besides from the results of tracers we can see that for every IP, the address stride of which may be not the same, so it means a bigger cost may not ensure a good performance on prefetching. Stream prefetching also needs a big hardware budget which faces the a problem with IP stride.

Table 2 shows the IPC of tagged_cache_stride prefetcher under each condition. Each of these three slows down the speed of prefetching and low bandwidth gives the most speed down. It makes sense because low bandwidth not only influences the time to fetch the cache miss but also slows down the speed of prefetching which means prefetching may not cover the memory latency.

I guess the bottleneck is the conflicts between the prefetching accuracy with the states we can record(hardware budget). This hypothesis comes from a simple idea that is if we can record all the information related to an address which contains the address to fetch before it and after it, is there a branch before it or will it go through a branch after execution and so on. The more information we get, the better pre things we can do (like prediction and prefetching). So in a limited hardware we need to record the most important information related to the address and find a good algorithm.