Efficient Victim Mechanism on Sector Cache Organization

Chunrong Lai [1] and Shih-Lien Lu [2]

[1] Intel China Research Center, 8F, Raycom Infotech Park A, No.2 Kexueyuan South Road ZhongGuanCun, Haidian District, Beijing China, 100080
chunrong.lai@intel.com
[2] Microprocessor Research, Intel Labs
shih-lien.l.lu@intel.com

Abstract. In this paper we present an victim cache design for caches organized with line that contains multiple sectors (sector cache). Sector caches use less memory bits to store tags than non-sectored caches. Victim cache has been proposed to alleviate conflict misses in a lower associative cache design. This paper examines how victim cache can be implemented in a sector cache and proposes a further optimization of the victim buffer design in which only the tags of the victim lines are remembered to re-use data in the sector cache. This design is more efficient because only an additional "OR" operation is needed in the tag checking critical path. We use a full system simulator to generate traces and a cache simulator to compare the miss ratios of different victim cache designs in sector caches. Simulation results show that this proposed design has comparable miss ratios with designs having much more complexity.

1 Introduction

In a cache an address tag (or tag) is used to identify the memory unit stored in the cache. The size of this memory unit affects how well a cache functions. For a fixed size cache larger unit size needs less memory bits to store tags and helps programs that possess special locality. However, larger unit may cause fragmentation making the cache less efficient when spatial locally is not there. Moreover, transferring each unit from lower memory hierarchy takes higher bandwidth. Smaller unit size allows more units to be included and may help programs that spread memory usage.

Sector cache[1][2] has been proposed as an alternative to strike a balance of cache unit sizes. A sector cache's memory unit is divided into sub-sections. Each unit needs only one tag thus saves tag memory bits. These sub-sections of a sector cache need not to be simultaneously brought in the cache allowing lower transferring bandwidth. Another advantage of sector caches is observed for multiprocessors systems because they reduce false sharing[3][4]. Sector cache's advantage is evident in that many microprocessors employ sector caches in their designs. For example, Intel's Pen-

P.-C. Yew and J. Xue (Eds.): ACSAC 2004, LNCS 3189, pp. 16–29, 2004.

tium® 4[1][5], SUN's SPARC™[6] and IBM's POWERPC™ G4™[7]/G5™[8] all employ sector cache in their cache organization.

This work intends to propose and evaluate further optimization techniques to improve performance of a sector cache. One of those designs is the victim cache[9]. A victim cache includes an additional victim buffer. When a line is replaced it is put into this small buffer which is full associative instead of just being discarded. The idea is to share the victim buffer entries among all sets since only a few of them are hotly contended usually. First, we discuss how victim buffer/cache idea can be applied in a sector cache. We evaluate two implementations of victim cache. One is called "line-victim" and the other is "sector-victim". We further propose a third victim mechanism design named "victim sector tag buffer"(VST buffer) for further utilize the sector cache lines. This design tries to address a sector cache's potential disadvantage of having larger unit size and could be under-utilized.

Since there are many different names[3][6][10][11][12][13][14][15] used to describe the units used in a sector cache, we first describe the terminology used in this paper. In our terminology a cache consist of lines which have tags associated with each of them. Each line consists of sub-units which are called sectors. This naming convention is the same as described in the manuals of Pentium® 4[5] and POWERPC™[7][8]. An example 4-way set-associative cache set is shown in figure 1. A valid bit is added to every sector to identify a partial valid cache line. We also use the terminology s-ratio which is defined as the ratio between the line size and the sector size. A sector cache with s-ratio equals to p is called p-sectored cache as [1]. The example in figure 1 it is a 4-sectored cache.

Address tag		Cache data
A	V	Sector 0(to A+SL)
Line 1	V	Sector 1(to A+2*SL)
	V	Sector 2(to A+3*SL)
	V	Sector 3(to A+4*SL)
B	V	Sector 0(to B+SL)
Line 2	V	Sector 1(to B+2*SL)
	V	Sector 2(to B+3*SL)
	V	Sector 3(to B+4*SL)

Address tag		Cache data
C	V	Sector 0(to C+SL)
Line 3	V	Sector 1(to C+2*SL)
	V	Sector 2(to C+3*SL)
	V	Sector 3(to C+4*SL)
D	V	Sector 0(to D+SL)
Line 4	V	Sector 1(to D+2*SL)
	V	Sector 2(to D+3*SL)
	V	Sector 3(to D+4*SL)

Fig. 1. Principles of sectored cache

This paper is organized as follows. In this section we introduce the concept of sector cache and victim mechanism. In the next section we first review other related works in this area. We then describe in more detail of our design. In section three we present the simulation methodology. In section four and five we introduce our simulation results on different cache levels. Finally we conclude with some observations.

[1] Pentium is a registered trademark of Intel Corp. or its subsidiaries in the United States and other countries.

2 Sector Cache with Victim Buffer

2.1 Related Work

Sector caches can be used to reduce bus traffic with only a small increase in miss ratio[15]. Sector cache can benefit in two-level cache systems in which tags of the second level cache are placed at the first level, thus permitting small tag storage to control a large cache. Sector cache also is able to improve single level cache system performance in some cases, particularly if the cache is small, the line size is small or the miss penalty is small. The main drawback, cache space underutilization is also shown in [13].

Rothman propose “sector pool” for cache space underutilization[13]. In the design, each set of set-associative cache compose of totally s-ratio sector lists. Each list has a fix number of sectors that the number is less than the associativity. S-ratio additional pointer bits, associate with a line tag, point to the actual sector as the index of the sector list. Thus a physical sector can be shared in different cache lines to make the cache space more efficient. Unlike our victim mechanism who tries to reduce the cache Miss ratio, this design more focus on cache space reduction. It depends on a high degree set associative cache. The additional pointer bits and the sector lists will make the control more complex. For example, the output of tag comparison need to be used to get the respond pointer bit first then can get the result sector. This lengthens the critical path. Another example is that different replacement algorithms for the cache lines and sector list need to be employed at the same time.

Seznec propose “decoupled sectored cache”[1][11]. A [N,P] decoupled sectored cache means that in this P-sectored cache there exists a number N such that for any cache sector, the address tag associated with it is dynamically chosen among N possible address tags. Thus a log2N bits tag, known as the selection tag, is associated with each cache sector in order to allow it to retrieve its address tag. This design increases the cache performance by allow N memory lines share a cache line if they use different sectors that some of the sectors have to be invalid at normal sector cache design. Our concern about this design is that the additional tag storage, say N-1 address tags and s-ratio * log2N selection tags for each line, need large amount of extra storage. Seznec himself use large(32 or 64) s-raio, which will make the validity check and coherence control very complex, to reduce tag storage before decoupling. We tried to implement this idea and saw the line-fill-in and line-replacement policy is important for the performance. If with a line-based-LRU-like fill-in/replacement policy proposed by ourselves, since Seznec did not give enough details of his policies, the decoupled sector cache will not perform better than our VST design, if with similar extra storage, given s-ratio range of 2~8. And, an additional compare need to be performed to the retrieved selection tag to ensure the sector data is right corresponded to the address tag which causes the tag matching. This also lengthens the tag checking critical path.

Victim caching was proposed by Jouppi[9] as an approach to reduce the cache Miss ratio in a low associative cache. This approach augments the main cache with a small fully-associate victim cache that stores cache blocks evicted from the main

cache as a result of replacements. Victim cache is effective. Depending on the program, a four-entry victim cache might remove one quarter of the misses in a 4-KB direct-mapped data cache. Recently [16] shows that victim cache is the most power/energy efficient design among the general cache misses reduction mechanisms. Thus it becomes a more attractive design because of the increasing demand of low power micro-architecture.

2.2 Proposed Design

In order to make our description clearer, we define several terms here. We call a reference to a sector cache a *block-miss* if the reference finds no matching tag within the selected set. We call a reference a *data-miss* if the tag matches but the referenced word is in an invalid sector. Thus a *miss* can be either block-miss or data-miss. Similar to [1][11] describe, for a P-sectored cache we divide the address A of a memory block in four sub-strings (A3, A2, A1, A0) defined as: A0 is a log2SL bit string which SL is the sector length, A1 is a log2P bit string show which sector this block is in if it is in a cache line, A2 is a log2nS bit string which nS is the number of the cache sets, A3 consists of the remaining highest significant bits. The main cache line need store only the bits in A3. Figure 2 show tag checking of directed-mapped case. A2 identify the only position of the tag to be compared in the tag array. (A2, A1) identify the only position the data sector can be. The data can be fetched without any dependency on the tag comparison. The processor pipeline may even start consuming the data speculatively without waiting for the tag comparison result, only roll back and restart with the correct data in the case of cache miss which is rarely happened, as line prediction.

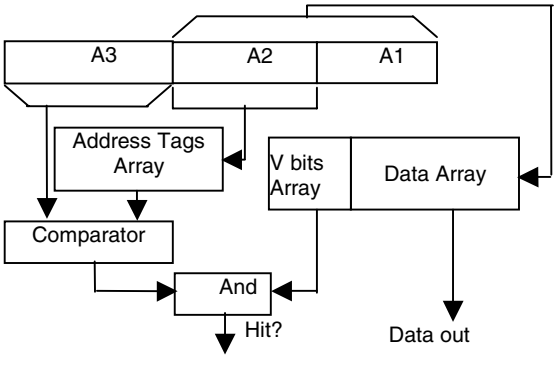

Fig. 2. Directed mapped sector cache tag checking

In the case of set-associative cache, (A2, A1) can only select the conceptual "sector set", then waiting for the comparison result of the address tags to get a line ID to deliver the correspond sector. Figure 3 is such an example of a 2-way associate sector

cache. In a lower-associate cache the sector data and the valid bits being select can be got independently with the tag comparison.

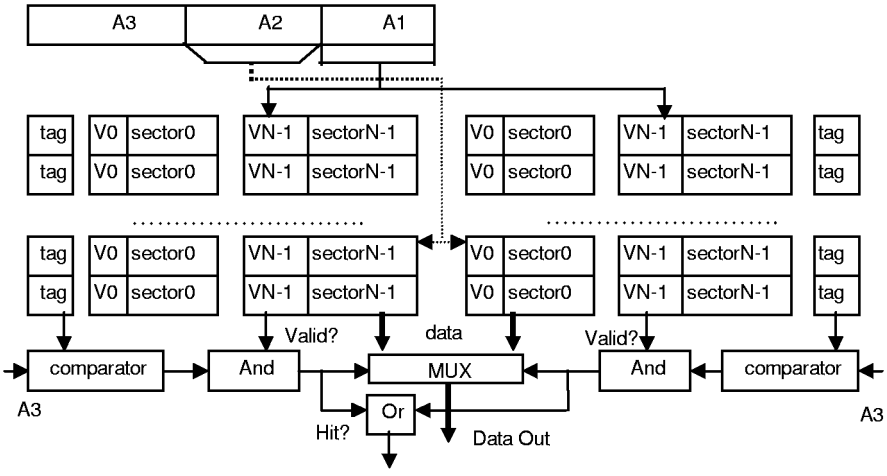

Fig. 3. 2-way associate sector cache tag checking

In figure 3 a line ID is needed, in critical path, as selection signal of the MUX. Line ID is founded after the tag comparison result. For a higher associate cache like a CAM, where a simple MUX may not be used, the data and valid bits could be not right at hand immediately. But Line ID retrieving still dominates in the critical path there[17].

As mentioned by many researchers victim cache can reduce cache Miss ratio. There are two straightforward victim designs for sector cache. One is line-victim cache(LVC), the other is sector-victim cache(SVC). Figure4 show their tag checking. Tag checking of the line victim cache is in the left and the other is in the right. The most difference between them is the data unit associate with the victim tag. In line-victim cache, the data unit is a cache line. And the data unit is a sector in the sector-victim cache. Thus the lengths of the victim tags of LVC and SVC are different. For same entries number LVC can be expected more cache misses saved due to more storage there, where SVC can be expected a little faster tag checking and data retrieve. Figure 4 do not connect the victim cache with main cache to avoid unnecessary complexity and allow architects to decide if swap the victim data with the main cache data when hit victim cache.

Both line-victim cache and sector-victim cache are paralleled accessed with the main sector cache. A cache line is evicted to the line- victim cache in case of cache replacement happens. As to the sector-victim cache, only the valid sectors in the whole line are evicted to the sector-victim-cache. Also when a new line is brought into cache, the sector-victim cache is checked to see if there are other sectors in the same line. If so the victim sectors are also brought into the main cache line to maintain a unique position of a cache line.

We know some of the requested data may still be in the data cache but it is just inaccessible because it has been invalidated. This paper describes another approach, called “VST buffer” to remember what is still in the data cache.

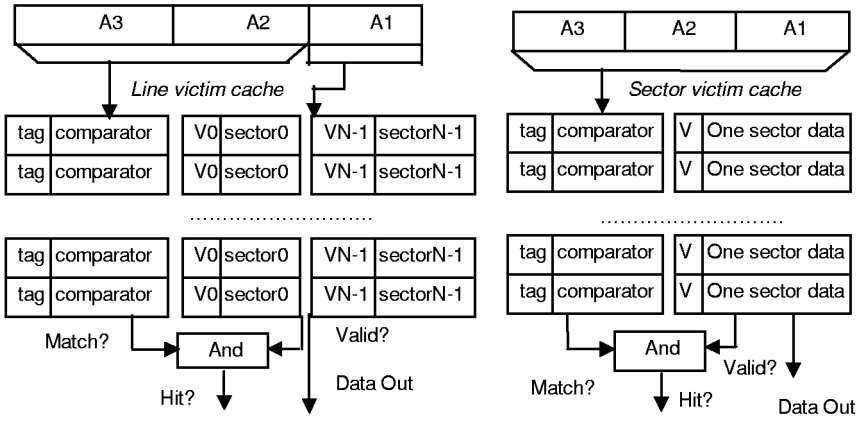

Fig. 4. Tag checking of line-victim cache and sector-victim cache

When a block miss happens and the set is full, a cache line must be replaced. Each sector of the replaced line will be mark as invalid. A new sector will be brought into the replaced line, and the cache tag will be updated. Thus some of the previously replaced line’s sector data may still be in the data array since not all sector, of the newly brought in cache line, is brought in. Only their valid bits are marked invalid. VST buffer is used to keep track of these sectors whose data is still the data array. Thus a VST entry consists of the victim tag, the victim valid bits and the “real location” line ID in the cache set. For a directed mapped main cache the line ID field is needless. The left side of figure 5 shows the VST buffer tag checking with a directed-mapped main cache. As seen from the figure the VST buffer produce an additional “VST hit” signal to be perform “or” operation with the main cache hit signal in the critical path, without affecting the sector fetching and consuming. In either a VST hit or a main cache hit the data can be processed continuously.

The right side of figure 5 shows the VST buffer tag checking of a set-associate main cache. A VST hit not only leads to a hit result but also deliver a line ID to the main cache selector to get the result data. This line ID signal is performed “or” operation with the main cache line ID signal to select final line. One extra cost here is when a new sector is brought into the main cache, if a data miss happens, the VST needs to be checked if the position contains a sector being victimized. If so the victim entry is invalidated or thrashed. This does not increase cache-hit latency since it happens when cache miss. Since there is already cache miss penalty the additional cost seems to be acceptable.

When compare the cost of the three victim mechanism connected with a p-Sectored Cache all compose of N entries. We see beside the similar comparators and the control, the line- victim cache need N line tags, data of N * line size and N*P

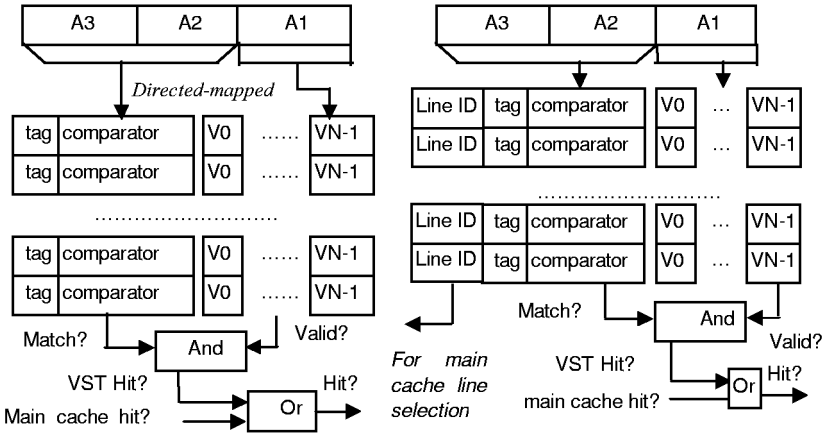

Fig. 5. Tag checking of Victim Sector Tag Buffer (VST buffer) with sector cache

valid bits; the sector-victim cache need N sector tags (each of it is log2P bits longer than a line tag), data of N * sector size and N valid bits; and the VST buffer need N line tags, N * P valid bits and N * log2Assoc bits of line IDs which Assoc is the cache associativity. So the line victim cache needs most resource among them as VST buffer need least resource.

In MP system, where the sector cache is proved efficient, there need additional cache coherence protocol, like MESI, to maintain the cache coherence. We think the victim mechanism will make the MP sector cache coherence protocol more complex. But we will not discuss the details here since it is beyond this paper's scope.

3 Simulation Methodology

Several SpecCPU2K[18] benchmarks (compiled with compiler option "–O3 –Qipo"), Java workload SpecJBB2K[19] with Java runtime environment JSEV1.4.1 which is an integer benchmark, and two commercial-like floating-point benchmarks, one is a speech recognition engine[20], the other is an echo cancellation algorithm[21] in audio signal processing, are used in our study.

In order to consider all the effects, including system calls, we use a full system simulator to generate memory reference traces. The simulator used is called SoftSDV[22]. The host system runs Windows 2000 and the simulated system is WindowsNT in batch mode using solid input captured in files. Then we run the traces through a trace-driven cache simulator.

We generate both L1 memory reference traces and L2 memory reference traces. After 20 billion instructions after system start up (the target application is configured auto-run in the simulation) we collect 200 million memory references as our L1 traces. We use 100 million references of them to warm up L1 cache and analysis the behavior of the latter 100 million. The L1 sector cache we simulated is mainly configured as below with small varieties: 16KBsize, 64B line size, 16B sector size, 4 way associ-

ate, LRU replacement algorithm and write-back approach. For L2 cache behavior we use a built-in first level cache together with trace generation. We warm up the built-in cache with 1 billion instructions. Then we collect L2 traces consist of 200 million read references. Also in our simulation we use 50 million L2 references to warm up the L2 cache. The hierarchy consist L2 sector cache we simulated is mainly configured as below with small varieties: L1: 16KBsize, 32B line size, 4-way associate, LRU replacement algorithm and write-back approach. L2: 1MB size, 128 byte line size, 32 byte sector size, 8-way associate, LRU replacement algorithm and write-back approach.

4 Level 1 Sector-Cache Simulation Results and Discussion

We present the L1 simulation data as the Miss Ratio Improvement Percentage (MRIP) of all benchmarks. The reason that we present L1 data first is that it is easier to correlate the observed L1 behavior back with the source code. Figure 6,7 are the MRIP trends with various parameters as the variable. All the numbers are computed as the geometric means of the different workload data also list in the paper. Figure 6 indicates that with larger number of the victim mechanism entries the miss ratio improvement increases. Since VST requires no data array we can implement a much larger victim buffer at the same cost of a smaller SVC/LVC and achieve the same (or even better) performance improvement. For example 128 entries VST performs comparably with 64 entries SVC or 32 entries LVC. Figure 6 also explores the improvement with several sector cache line sizes and sector sizes. We observed that VST performs better with larger s-ratios. This is because of higher underutilization cache space exist with higher s-ratio. On the other hand SVC and LVC performs better with larger line and sector sizes. Figure 7 compares how the victim mechanisms affect caches with different associativities or different cache sizes. It is not surprising to learn that all three forms of victim mechanisms help the lower associative cache better. This is because higher associativity already reduced much of the conflict misses victim cache is targeting. It is also seen smaller L1 cache benefits more from the victim mechanisms. As frequency of microprocessors continues to grow, smaller but faster (lower associativity gives faster cache too) cache will be more prevalent.

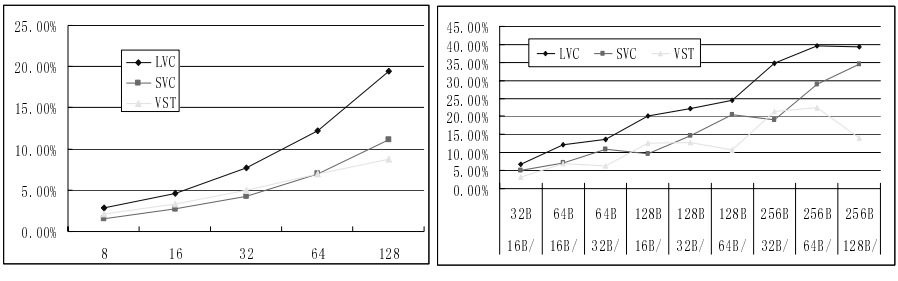

Fig. 6. MRIP with victim entries or line/sector sizes (higher is better)

We observe that LVC gives the best Miss ratio improvement at the highest hardware cost. While the SVC approach we used for this study needs the second highest hardware cost, it is not better than VST approach. The VST approach is a reasonable approach in terms of hardware design complexity and overhead.

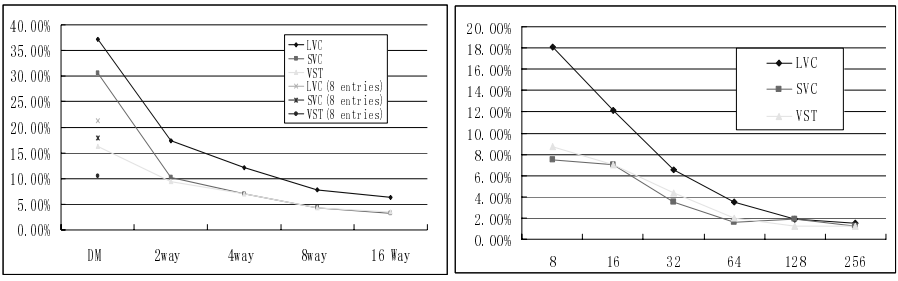

Fig. 7. MRIP with different associativities or L1 sizes

The cache miss ratios with different number of victim entries, correspond to the left figure of figure 6, are listed in table 1. The data of other figures are listed in appendix. Table 1 also list corresponding block misses ratios for further investigation.

Table 1. Miss Ratios and Block Miss Ratios with numbers of victim entries

L1 victim		Miss ratios						Block Miss ratios					
entries		8	16	32	64	128	origin	8	16	32	64	128	origin
LVCS	LVC	2.77	2.73	2.69	2.64	2.57	2.92	1.22	1.20	1.17	1.14	1.10	1.36
	SVC	2.79	2.76	2.73	2.69	2.64	2.92	1.25	1.23	1.20	1.18	1.15	1.36
	VST	2.81	2.78	2.76	2.73	2.71	2.92	1.25	1.23	1.20	1.18	1.16	1.36
AMMP	LVC	9.01	8.94	8.81	8.58	8.25	9.08	3.88	3.85	3.77	3.63	3.42	3.91
	SVC	9.04	9.00	8.93	8.77	8.50	9.08	3.90	3.88	3.85	3.77	3.65	3.91
	VST	9.02	8.96	8.86	8.71	8.56	9.08	3.87	3.83	3.76	3.67	3.57	3.91
MESA	LVC	0.63	0.57	0.56	0.54	0.51	1.67	0.27	0.21	0.20	0.19	0.18	0.95
	SVC	0.72	0.65	0.58	0.55	0.54	1.67	0.35	0.29	0.23	0.20	0.20	0.95
	VST	0.99	1.01	1.02	1.03	1.02	1.67	0.49	0.49	0.49	0.48	0.48	0.95
SAEC	LVC	3.29	3.26	3.19	3.07	2.84	3.35	0.93	0.92	0.90	0.86	0.80	0.95
	SVC	3.34	3.32	3.29	3.25	3.18	3.35	0.94	0.94	0.93	0.92	0.90	0.95
	VST	3.33	3.32	3.30	3.27	3.25	3.35	0.94	0.93	0.92	0.91	0.90	0.95
GZIP	LVC	10.67	10.56	10.37	10.04	9.45	10.81	7.44	7.33	7.13	6.76	6.13	7.58
	SVC	10.68	10.58	10.41	10.12	9.63	10.81	7.47	7.39	7.23	6.97	6.55	7.58
	VST	10.69	10.65	10.45	10.13	9.70	10.81	7.45	7.33	7.12	6.75	6.14	7.58
GCC	LVC	2.32	2.27	2.09	1.76	0.88	2.35	0.68	0.67	0.62	0.53	0.29	0.69
	SVC	2.34	2.33	2.31	2.23	2.02	2.35	0.69	0.68	0.68	0.66	0.60	0.69
	VST	2.30	2.24	2.14	2.07	2.06	2.35	0.67	0.66	0.62	0.60	0.59	0.69
SJBB	LVC	3.88	3.83	3.76	3.67	3.51	3.96	1.63	1.60	1.56	1.52	1.45	1.67
	SVC	3.90	3.88	3.84	3.77	3.69	3.96	1.65	1.63	1.61	1.57	1.53	1.67
	VST	3.91	3.88	3.84	3.78	3.71	3.96	1.65	1.62	1.59	1.55	1.50	1.67

As shown in table 1, the benchmark “mesa” got most of cache misses reduction with victim mechanism regardless LVC/SVC, or VST we used. “ammp” got least misses reduction with LVC and “saec” got least misses reduction with SVC and VST.

For the workload “mesa”, we observed the block Miss ratio reduce much more significantly with victim mechanism compared to the cache Miss ratio. Thus with

victim mechanism the workload basically keeps more cache lines to save cache misses in this level. Other issues, like quantitative spatial localities that make SVC performs differently, say reduces different percentage of miss ratio reduced by LVC with same entries, play minor role in this level.

In some cases (GCC with 8 victim entries), the VST buffer approach performs better than LVC even without any data array. After investigation we concluded that the VST buffer approach sometimes uses the victim buffer more efficiently and can avoid be thrashed. Victim cache contains data that may be used in future. But the data can also be kicked out of the victim cache before it is needed. For example, streaming accesses, if miss the main cache, will evict main cache lines to update the victim cache. Thus the victim cache gets thrashed and may lost useful information. It plays differently in VST approach. We see in non-sector cache, streaming accesses are mapping in different sets of cache which make it difficult to be detected. In a sector cache the next sector of a cache is inherently subsequence of the previous sector. Figure 8 shows the VST states with one by one streaming accesses (or sequential) going to the cache, only one VST entry is enough handling them since the entry can be re-used(disabled) after a whole main cache line fill-in. Thus the whole buffer will keep longer history. This is right the case VST performs better than LVC for GCC.

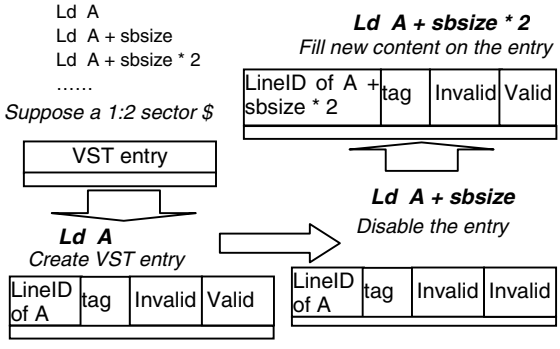

Fig. 8. Avoid be thrashed by streaming access

5 Level 2 Sector-Cache Simulation Results and Discussion

We also explore the possibility of applying our proposed methods on level-two cache design. This time only those references that missed the build-in level one cache are collected in the trace file. Table 2 illustrates the tabulated result in terms of miss ratio for various entries. Data with other parameters are also listed in appendix.

There are several observations made from the L2 data. First, LVC performs better than SVC with same entries but worse than SVC with s-ratios, here 4 times, of entries, same as be observed from L1 data. Second, in lower level set-associative cache, victim mechanism performs differently as L1. It does not save so many cache misses as

Table 2. L2 Miss Ratio with Victim Mechanism

Entries		16	32	64	128	256	Origin
Ammp	LVC	15.36	15.06	14.52	13.71	12.85	15.62
	SVC	15.49	15.34	15.06	14.58	13.89	15.62
	VST	14.82	14.24	13.71	13.30	12.91	15.62
LVCSR	LVC	36.57	36.53	36.46	36.33	36.06	36.60
	SVC	36.59	36.57	36.54	36.49	36.38	36.60
	VST	36.57	36.54	36.48	36.36	36.14	36.60
Mesa	LVC	9.64	9.64	9.63	9.63	9.63	9.64
	SVC	9.64	9.64	9.64	9.64	9.63	9.64
	VST	9.64	9.64	9.64	9.64	9.64	9.64
Gcc	LVC	8.52	8.51	8.50	8.48	8.44	8.52
	SVC	8.52	8.52	8.51	8.50	8.49	8.52
	VST	8.51	8.51	8.49	8.47	8.45	8.52
Gzip	LVC	0.33	0.33	0.33	0.33	0.33	0.33
	SVC	0.33	0.33	0.33	0.33	0.33	0.33
	VST	0.33	0.33	0.33	0.33	0.33	0.33
Mcf	LVC	50.47	50.43	50.36	50.23	49.98	50.50
	SVC	50.48	50.47	50.44	50.38	50.26	50.50
	VST	50.47	50.45	50.40	50.35	50.24	50.50
SAEC	LVC	0.40	0.38	0.37	0.37	0.37	0.41
	SVC	0.41	0.40	0.39	0.38	0.37	0.41
	VST	0.42	0.42	0.42	0.42	0.42	0.41

L1 cache. This is not surprising since a small L1 already catch a significant part of data locality and L2 reference patterns tend to be more irregular. Third, the VST buffer performs well among the three victim mechanisms in this memory hierarchy level. It can outperform LVC and SVC for the benchmark “ammp”. Even it is more difficult to correlate the L2 references back with the source or binary, than L1 references. We still ascribe the better VST performance to its property of avoiding be thrashed. As to the workloads, “ammp” and “SAEC” get most significant cache misses reduction here. This behavior is opposite to the L1 behavior. Also the significant block miss reduction can not be observed in this level as the data in appendix shows. Thus we suggest that the extra storage of LVC and SVC benefit more from the general data locality; and VST benefit more from the cache underutilization whether the reference pattern is regular or not.

6 Conclusion

We have described three possible implementation of victim buffer design in a sector cache. They have different complexity and hardware overhead. Several up-to-date applications are used to evaluate their performance in terms of miss ratio. Overall three mechanisms have comparable cache misses reduction. For a directed-mapped Level 1 cache, the mechanisms can save significant amount of cache misses.

Among the three mechanisms LVC gives the best performance with highest overhead. Whether SVC is performance/cost effective or not rely on the quantitative spatial locality of the workload.

We also investigate several benefits of VST in this paper. Include the low-cost design, keeping longer victim history and be more able to capture irregular reference pattern in lower memory hierarchy.

Acknowledgement. We thank the AudioProcessing group and the OpenRuntimePlatform group of Intel China Research center for giving us their up-to-date workloads and providing helpful discussions on porting workloads to our simulator. We also thank Zhu Ning and Peter Liou for providing necessary computing infrastructure support.

References

[1] Andre. Seznec. "Decoupled sectored caches". IEEE Trans. on Computers, February, 1997

[2] D.A.Patterson, J.L.Hennessy, "Computer architecture: A quantitative approach", Morgan Kaufmann Publishers Inc., San Francisco,1996.

[3] Kuang-Chih Liu, Chung-Ta King, "On the effectiveness of sectored caches in reducing false sharing misses" International Conference on Parallel and Distributed Systems, 1997

[4] Won-Kee Hong, Tack-Don Han, Shin-Dug Kim and Sung-Bong Yang, "An Effective Full-Map Directory Scheme for the Sectored Caches", International Conference/Exhibition on High Performance Computing in Asia Pacific Region, 1997

[5] Hinton, G; Sager, D.; Upton, M.; Boggs, D.; Carmean, D.; Kyker, A.; Roussel, P., "The Microarchitecture of the Pentium® 4 processor", Intel Technology Journal, 1st quarter, 2001, http://developer.intel.com/technology/itj/q12001/articles/art_2.htm

[6] "UltraSPARC™ Iii User's Manual", Sun Microsystems, 1999

[7] PowerPC™ , "MPC7400 RISC Microprocessor Technical Summary ", Mororola, Order Number: MPC7400TS/D, Rev. 0, 8/1999

[8] Victor Kartunov, "IBM PowerPC G5: Another World", X-bit Labs, Jan. 2004 http://www.xbitlabs.com/articles/cpu/display/powerpc-g5_6.html

[9] N. Jouppi, "Improving direct-mapped cache performance by the addition of a small fully associative cache and prefetch buffers", International Symposium on. Computer Architecture 1990

[10] Jeffrey B. Rothman, Alan Jay Smith: "Sector Cache Design and Performance". International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2000

[11] Andre. Seznec. "Decoupled sectored caches: conciliating low tag implementation cost". International Symposium on. Computer Architecture, 1994

[12] J.S.Lipty. "Structural Aspects of the System/360 Model 85, Part II: The Cache. IBM Systems Journal, Vol. 7, 1968

[13] Jeffrey B. Rothman and Alan Jay Smith. "The Pool of SubSectors Cache Design". International Conference on Supercomputing, 1999

[14] Mark D. Hill and Alan Jay Smith. "Experimental Evaluation of On-Chip Microprocessor Cache Memories". International Symposium on Computer Architecture, June 1984

[15] James R. Goodman. "Using Cache Memory to Reduce Processor Memory Traffic". International Symposium on. Computer Architecture 1983

[16] G. Albera and R. Bahar, " Power/performance Advantages of Victim Buffer in High-Performance Processors", IEEE Alessandro Volta Memorial Workshop on Low-Power Design,1999
[17] Farhad Shafai, Kenneth J. Schultz, G..F. Randall Gibson, Armin G. Bluschke and David E. Somppi, "Fully Parallel 30-MHz, 2.5-Mb CAM", IEEE journal of solid-state circuits, Vol. 33, No. 11, November 1998
[18] SPEC CPU2000, http://www.specbench.org/osg/cpu2000
[19] SPEC JBB 2000, http://www.specbench.org/jbb2000
[20] C.Lai, S. Lu and Q. Zhao, "Performance Analysis of Speech Recognition Software", Workshop on Computer Architecture Evaluation using Commercial Workloads, International Symposium on High Performance Computer Architecture, 2002
[21] J. Song, J. Li, and Y.-K. Chen, "Quality-Delay and Computation Trade-Off Analysis of Acoustic Echo Cancellation On General-Purpose CPU," International Conference on Acoustics, Speech, and Signal Processing, 2003.
[22] R. Uhlig et. al., "SoftSDV: A Pre-silicon Software Development Environment for the IA-64 Architecture", Intel Technology Journal, 4th quarter, 1999. http://developer.intel.com/technology/itj/q41999/articles/art_2.htm

Appendix: More Simulation Data

Sector/ line size		16B/ 32B	16B/ 64B	32B/ 64B	16B/ 128B	32B/ 128B	64B/ 128B	32B/ 256B	64B/ 256B	128B/ 256B
LVCSR	LVC	2.52	2.64	1.68	2.70	1.72	1.17	1.74	1.18	0.87
	SVC	2.53	2.69	1.70	2.83	1.78	1.19	1.87	1.24	0.88
	VST	2.59	2.73	1.79	2.90	1.89	1.38	2.02	1.51	1.20
	ORI	2.65	2.92	1.92	3.23	2.15	1.55	2.86	2.19	1.69
AMMP	LVC	8.17	8.58	5.32	9.00	5.64	3.88	5.74	3.96	3.04
	SVC	8.23	8.77	5.37	9.36	5.76	3.90	6.12	4.12	3.06
	VST	8.27	8.71	5.46	9.09	5.72	4.02	5.90	4.16	3.40
	ORI	8.38	9.08	5.67	9.86	6.25	4.37	6.68	4.69	3.66
MESA	LVC	0.53	0.54	0.31	0.54	0.31	0.19	0.31	0.20	0.14
	SVC	0.54	0.55	0.32	0.63	0.33	0.20	0.53	0.25	0.15
	VST	0.95	1.03	0.75	1.08	0.78	0.69	0.86	0.75	1.39
	ORI	1.36	1.67	1.20	2.72	1.98	1.62	2.72	2.29	1.93
SAEC	LVC	3.11	3.07	1.60	2.95	1.54	0.83	1.45	0.78	0.45
	SVC	3.16	3.25	1.66	3.38	1.73	0.89	1.83	0.93	0.49
	VST	3.22	3.27	1.72	3.33	1.75	0.96	1.80	0.99	0.58
	ORI	3.25	3.35	1.75	3.50	1.83	0.99	1.98	1.08	0.62
GZIP	LVC	9.09	10.0	8.28	10.7	9.00	7.54	9.36	7.95	6.57
	SVC	9.10	10.1	8.29	11.0	9.16	7.59	10.0	8.38	6.76
	VST	9.18	10.13	8.46	10.86	9.26	8.02	9.83	8.66	7.97
	ORI	9.54	10.8	9.09	12.1	10.5	9.09	12.1	10.8	9.48
GCC	LVC	1.92	1.76	0.95	1.46	0.79	0.45	0.46	0.27	0.18
	SVC	2.05	2.23	1.09	2.49	1.19	0.57	1.36	0.63	0.31
	VST	2.04	2.07	1.15	2.08	1.16	0.69	1.17	0.69	0.45
	ORI	2.14	2.35	1.25	2.64	1.40	0.77	1.60	0.88	0.51
SJBB	LVC	3.55	3.67	2.31	3.72	2.34	1.54	2.36	1.55	1.08
	SVC	3.59	3.77	2.34	3.97	2.45	1.58	2.70	1.70	1.11
	VST	3.62	3.78	2.40	3.91	2.50	1.72	2.47	1.89	1.41
	ORI	3.70	3.96	2.52	4.33	2.79	1.89	3.34	2.83	1.68

L1 Data		DM	2way	4way	8way	16 Way	DM(8 Entries)	8KB	16KB	32KB	64KB	128KB	256KB
LVCSR	LVC	2.71	2.68	2.64	2.61	2.60	3.35	2.77	2.64	2.50	2.35	2.18	1.92
	SVC	2.75	2.71	2.69	2.64	2.63	3.70	2.90	2.69	2.52	2.36	2.18	1.93
	VST	4.16	3.10	2.73	2.63	2.62	4.33	3.01	2.73	2.52	2.35	2.18	1.92
	ORI	5.31	3.64	2.92	2.69	2.66	5.31	3.35	2.92	2.58	2.37	2.19	1.93
AMMP	LVC	9.11	8.76	8.58	8.51	8.49	9.76	9.47	8.58	7.92	7.64	7.44	6.20
	SVC	9.35	9.01	8.77	8.65	8.65	9.89	9.80	8.77	8.00	7.66	7.45	6.22
	VST	10.08	8.95	8.71	8.61	8.59	10.44	9.89	8.71	7.96	7.65	7.44	6.13
	ORI	11.36	9.38	9.08	8.94	8.94	11.36	10.27	9.08	8.10	7.68	7.45	6.23
MESA	LVC	0.56	0.54	0.54	0.54	0.55	4.10	0.63	0.54	0.41	0.30	0.16	0.10
	SVC	0.74	0.55	0.55	0.56	0.57	4.91	0.67	0.55	0.44	0.31	0.17	0.10
	VST	4.08	1.56	1.03	0.60	0.58	4.94	1.48	1.03	0.45	0.35	0.18	0.10
	ORI	6.68	3.02	1.67	0.66	0.60	6.68	3.22	1.67	0.47	0.32	0.17	0.11
SAEC	LVC	3.27	3.15	3.07	3.01	3.02	4.09	3.82	3.07	2.27	1.20	1.00	0.64
	SVC	3.76	3.37	3.25	3.23	3.23	4.26	4.19	3.25	2.39	1.31	1.01	0.65
	VST	4.22	3.43	3.27	3.24	3.23	4.38	4.38	3.27	2.31	1.29	1.00	0.64
	ORI	4.72	3.56	3.35	3.32	3.32	4.72	4.59	3.35	2.43	1.35	1.01	0.65
GZIP	LVC	10.30	10.13	10.04	10.00	9.97	11.53	11.27	10.04	8.08	5.05	1.81	0.37
	SVC	10.35	10.21	10.12	10.07	10.04	11.50	11.35	10.12	8.19	5.19	1.91	0.38
	VST	14.62	10.30	10.13	10.07	10.03	15.51	11.58	10.13	8.13	5.05	1.83	0.38
	ORI	16.32	11.17	10.81	10.68	10.62	16.32	12.90	10.81	8.65	5.47	2.02	0.39
GCC	LVC	1.55	1.69	1.76	1.82	1.82	1.98	3.82	1.76	0.32	0.21	0.17	0.15
	SVC	1.88	2.17	2.23	2.24	2.19	2.00	4.46	2.23	0.37	0.22	0.17	0.15
	VST	1.86	2.00	2.07	2.19	2.23	2.05	4.32	2.07	0.36	0.22	0.18	0.15
	ORI	2.30	2.26	2.35	2.43	2.47	2.30	4.47	2.35	0.49	0.23	0.19	0.15
SJBB	LVC	3.76	3.71	3.67	3.67	3.68	4.38	4.05	3.67	3.19	2.58	2.00	1.76
	SVC	4.07	3.89	3.77	3.75	3.75	4.56	4.42	3.77	3.26	2.63	2.03	1.76
	VST	4.56	3.98	3.78	3.73	3.71	4.86	4.42	3.78	3.25	2.63	2.06	1.78
	ORI	5.37	4.23	3.96	3.84	3.80	5.37	5.00	3.96	3.32	2.66	2.04	1.77

L2 Data		256KB	512KB	1MB	2MB	4MB	8MB	DM	2way	4way	8way	16way
Ammp	LVC	55.76	38.38	13.71	8.21	7.36	2.28	35.49	33.31	19.93	13.71	12.59
	SVC	56.64	39.05	14.58	8.27	7.41	2.31	36.26	34.04	20.67	14.58	13.00
	VST	54.37	36.69	13.30	8.24	7.39	2.27	36.13	33.16	18.76	13.30	12.59
	ORI	57.57	40.07	15.62	8.49	7.58	2.34	38.18	35.11	21.65	15.62	13.75
LVCS	LVC	52.63	46.03	36.33	26.49	20.72	14.66	40.69	37.79	36.65	36.33	36.08
	SVC	52.89	46.21	36.49	26.57	20.75	14.68	41.31	38.00	36.83	36.49	36.23
	VST	52.81	46.11	36.36	26.51	20.72	14.67	41.32	37.88	36.70	36.36	36.10
	ORI	53.28	46.40	36.60	26.63	20.76	14.69	42.29	38.19	36.96	36.60	36.35
SAEC	LVC	14.86	0.54	0.37	0.36	0.34	0.32	7.16	0.52	0.38	0.37	0.36
	SVC	15.63	0.82	0.38	0.36	0.34	0.32	7.64	0.64	0.45	0.38	0.36
	VST	14.70	0.87	0.42	0.36	0.34	0.32	7.45	0.76	0.45	0.42	0.36
	ORI	15.78	1.07	0.41	0.36	0.34	0.32	8.48	0.95	0.49	0.41	0.36