CSCE 614 – COMPUTER ARCHITECTURE TERM PROJECT

SUBMITTED BY
VAMSI KRISHNA KODATI
&
RAVI SANKAR RAJU

Vamsi Krishna Kodati
UIN: 92300375

Ravi Sankar Raju
UIN: 722007134

LINE DISTILLATION: INCREASING CACHE CAPACITY BY FILTERING UNUSED WORDS IN CACHE LINES

1. ABSTRACT

Caches are organized as lines to exploit spatial locality. However in cases where spatial locality is low, majority of the words in least recently accessed line are unused. They occupy cache space and do not contribute to any cache performance. The Line Distillation (LDIS) method in [1] proposes a novel approach of filtering unused words. The extra cache space created by this method is used to retain used words of evicted lines. This improves cache performance by reducing conflict misses. To employ (LDIS), a new cache called distill cache with word-size granularity is proposed in [1]. We employed distill cache and ran experiments on SPEC CPU2000 benchmarks to show LDIS can lead to significant improvement in cache performance.

2. INTRODUCTION

Caches are organized as lines to exploit spatial locality. Spatial locality varies across memory accesses in an application and different applications. When spatial locality is more, bigger cache line improves cache performance. When spatial locality is low, bigger cache lines do not improve any cache performance. The unused words block cache space which otherwise can be used to store words/lines frequently used. For efficient use of cache space a novel LDIS approach proposed in [1] is used. To employ LDIS, a novel cache organization Distill Cache proposed in [1] is used. Distill Cache consists sub caches, Line Organized Cache (LOC) and Word Organized Cache (WOC). Cache blocks are initially placed in LOC. When the least recently used line is evicted, used words of evicted line are placed in corresponding cache line of WOC. To prevent a newly evicted line from LOC from replacing several useful lines in WOC, median-threshold filtering proposed in [1] is employed. The evicted LOC line is placed in WOC only when the number of used words in evicted line is less than the median-threshold value of WOC. This improves cache performance of distill cache by retaining useful lines.

We have implemented Distill cache employing LDIS with and without median threshold filtering at L2 level caches in simplescalar simulator. Its performance against various SPEC CPU2000 benchmarks is studied. We refrain from employing distill cache at L1 level. Cache access time or hit latency in Distill Cache will be relatively higher than normal set associative cache because of word level access. Since faster hit time is critical to L1 caches, Distill Cache is avoided at this level. Also associativity of L1 caches are low. Low associative distilled caches can tend to show decreased performance because it aggressively evicts unused words in a cache line and it can result in more word-misses. The likelihood of unused words not used again in least recently used cache line increases as associativity increases. L2 caches have higher associativity and small increase in hit-time like additional one cycle does not affect cache performance significantly. L2 cache performance is significantly affected by miss rate than faster hit time, hence distill caches are more suitable at L2 level.

The performance of 512KB, 64B line-sized 8-way set associative distill cache is evaluated against 512KB, 64B line-sized 8-way set associative cache and 1MB, 64B line-sized 8-way set associative cache.

3. Implementation

3.1 Tracking the word usage

To track the used words in a line, each line is given extra bits where each bit corresponds to a word in the line, which we will refer to as footprint. Bits that are 1 represent words that are used, 0 represent unused words. When a line is replaced as a whole, each bit in the footprint is set to 0. Footprint keeps track of used words in a line.

3.2 Distil Cache Organization

In an n-way set associative distill cache, (n-1) ways correspond to LOC and the nth way correspond to WOC. LOC cache is like any other cache with line size granularity. Cache accesses in LOC are similar to conventional caches. WOC is a word size granular cache. Cache access in WOC is word oriented. Hence each valid word in WOC has a tag-entry called word-tag associated with it. Multiple used words of a cache line are always stored consecutively in WOC line. Only multiple words in order of power two (1, 2, 4, 8 or 16) can be stored in WOC line. This is done so that multi-word accesses in WOC are aligned. For eg, two used words of a LOC line are stored either in positions (0,1) or (2,3) or (4,5) or (6,7) in 8 word WOC line. In WOC cache line, along with word-tags additional bits called word valid bits and dirty bits are added for each word to determine if it is valid or not, dirty or not. 1's in the valid bits determine the word is valid, 1's in dirty bits determine the word is dirty.

3.3 Hit and Miss

Different scenarios that arise due to hit and miss when distill cache is employed are discussed below.

3.3.1 L1 Hit

In this case, the footprint bit that corresponds to the word that is accessed is set to 1 and the word is sent to the processor.

3.3.2 Eviction from L1

In cases where L1 data line has to be replaced with a new block, the old block is moved to L2 where, if the block already exists, the footprint is updated.

(footprint from L1) OR (footprint in L2)

If there's no existence of the block, place the block into a new line and evict any existing line. While the eviction happens, use the footprint and threshold to determine which words go into the WOC.

3.3.3 L1 miss and LOC Hit

There is a miss in L1 and the corresponding block can be found in LOC of L2, then the execution continues like normal cache. The data line is sent to L1.

3.3.4 WOC Hit

There is a miss in L1 and L2's LOC and hence WOC is accessed. WOC hit implies that the tag has been matched and the word's valid bit is 1. The cache line is sent to L1 along with the valid-bit vector which indicates which positions contain valid words

3.3.5 Hole Miss

There is a miss in L1 and L2's LOC and hence WOC is accessed. Hole miss implies that there is line-tag match in WOC of L2 but word-tag match fails i.e. the requested word is not in the cache. In this case, a request for this particular line is sent to memory which is used to update LOC and L1. If there are any dirty words, it is copied to LOC before sending the line to L1.

3.3.6 Line Miss

The word request misses in every path and hence fetched from memory. Another line from the available n-1 blocks is evicted to the distill cache and replaced with this new line.

3.4 Replacement Policy

When the WOC overflows, one of the multi-word LOC line stored in WOC line has to be replaced. Since multi-word lines are in order of powers of 2, replacement policy should support size variable replacement in powers of 2. The replacement should occur only at word-aligned boundaries. During eviction of a multi-word line all the words in it are evicted. Hence to identify start of a multi-line segment in WOC a head bit is maintained for each word. 1 in head bit indicates it is the head word of the line, 0 indicates following words of the line. Segments are chosen randomly for replacement.

3.5 Use of Threshold in Distillation

During eviction, when a LOC line has almost every word used, store these words in WOC may evict several useful lines from WOC. In other words, the number of different lines that are stored in the WOC reduces significantly. As this unique number of lines in a WOC tends to 1, it behaves as a regular block in the set and the benefit of cache distillation is defeated. To avoid this, we have a threshold and any line that has used words ore than the threshold is not placed in the WOC.

3.5.1 Choosing the Threshold

High Threshold nullifies the purpose of distillation and Low threshold means that there might not be a lot of entries in the WOC. If the threshold is set to 1, it means that if the line has more than a word used, it is not placed in WOC. To find an optimal threshold, solution proposed in [1] which takes median of the number of words used in each line at the time of eviction is implemented. We have 'n' unique counters to count number of lines that had 1 to 'n' used words at the time of eviction. We also maintain a eviction-sum counter to count the number of line being evicted. Now we sum counters 0 to 'n' until the sum reaches half the value of the eviction-sum counter. This gives the median value. This process is repeated for every 4k evictions.

4. EXPERIMENTS

4.1 Base L1 Cache Configuration Used

Type of Cache	Number of Sets	Associativity	Line Size (Bytes)	Cache Size (KB)	Distill Cache	Median Threshold
Instruction	64	2	64	8	No	No
Data	64	2	64	8	No	No

4.2 Varying L2 Cache Configuration Used

Type of Cache	Number of Sets	Associativity	Line Size (Bytes)	Cache Size (KB)	Distill Cache	Median Threshold
Unified	1024	8	64	512	No	No
Unified	1024	8	64	512	Yes	No
Unified	1024	8	64	512	Yes	Yes
Unified	2048	8	64	1024	No	No

4.3 Simplesim Simulator

4.3.1 New Cache Options added to simplesim simulator

-l2_distillcache (false/true) - When set to true, enable distillation.
-l2_distill_threshold (true/false) - When set to true, distillation is restricted by the threshold (number of used words). When set to false, distillation always occurs.

512KB Line Distilled Unified Cache with median threshold (LDIS-MT):

```
./simplesim-3.0/sim-outorder -max:inst 200000000 -fastfwd 20000000 -redir:sim benchmark_ldis_mt_output.txt -bpred 2lev -bpred:2lev 1 256 4 0 -bpred:ras 8 -bpred:btb 64 2 -cache:il1 il1:64:64:2:l -cache:dl1 dl1:64:64:2:l -l2_distillcache true -l2_distill_threshold true -cache:il2 dl2 -cache:dl2 ul2:1024:64:8:l
```

512KB Line Distilled Unified Cache without median threshold (LDIS):

```
./simplesim-3.0/sim-outorder -max:inst 200000000 -fastfwd 20000000 -redir:sim benchmark_ldis_mt_output.txt -bpred 2lev -bpred:2lev 1 256 4 0 -bpred:ras 8 -bpred:btb 64 2 -cache:il1 il1:64:64:2:l -cache:dl1 dl1:64:64:2:l -l2_distillcache true -l2_distill_threshold false -cache:il2 dl2 -cache:dl2 ul2:1024:64:8:l
```

512KB Unified Cache (Base 512KB):

```
./simplesim-3.0/sim-outorder -max:inst 200000000 -fastfwd 20000000 -redir:sim benchmark_ldis_mt_output.txt -bpred 2lev -bpred:2lev 1 256 4 0 -bpred:ras 8 -bpred:btb 64 2 -cache:il1 il1:64:64:2:l -cache:dl1 dl1:64:64:2:l -l2_distillcache false -l2_distill_threshold false -cache:il2 dl2 -cache:dl2 ul2:1024:64:8:l
```

1MB Unified Cache (Base 1MB):

./simplesim-3.0/sim-outorder -max:inst 200000000 -fastfwd 20000000 -redir:sim benchmark_ldis_mt_output.txt -bpred 2lev -bpred:2lev 1 256 4 0 -bpred:ras 8 -bpred:btb 64 2 -cache:il1 il1:64:64:2:l -cache:dl1 dl1:64:64:2:l -l2_distillcache false -l2_distill_threshold false -cache:il2 dl2 -cache:dl2 ul2:2048:64:8:l

4.4 Benchmarks

4.4.1 Integer Benchmarks

bzip2, crafty, gap, gcc, gzip, mcf, parser, twolf, vortex, vpr, bzip2

4.4.2 Floating Benchmarks

ammp, applu, apsi, art, equake, fma3d, galgel, parser, lucas, mesa, mgrid, swim

5. RESULTS AND ANALYSIS:

5.1 Variation of L2 miss rate with L2 cache configuration:

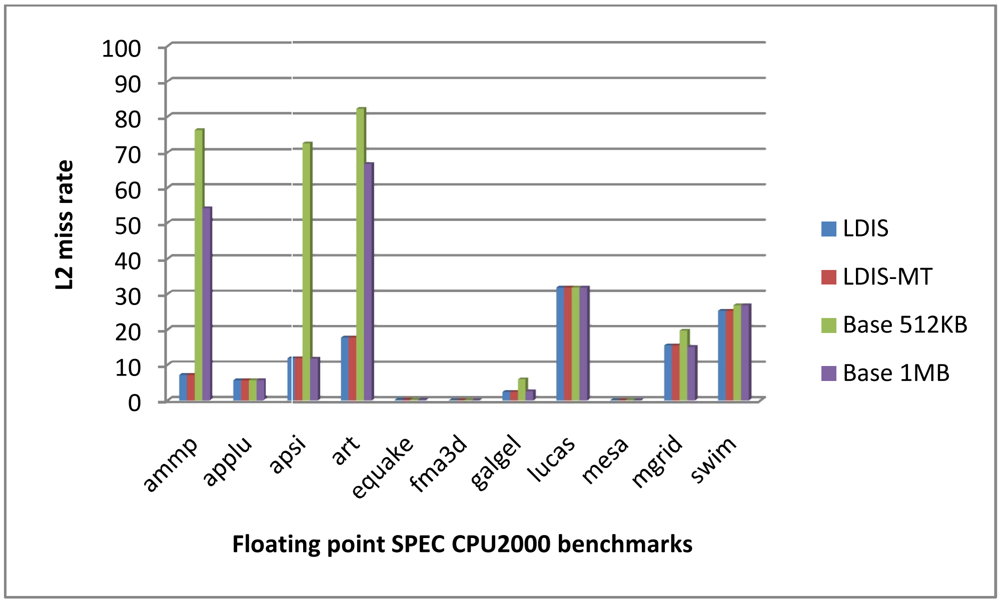

Fig 1: Variation of L2 miss rate with L2 cache configuration in Floating point benchmarks

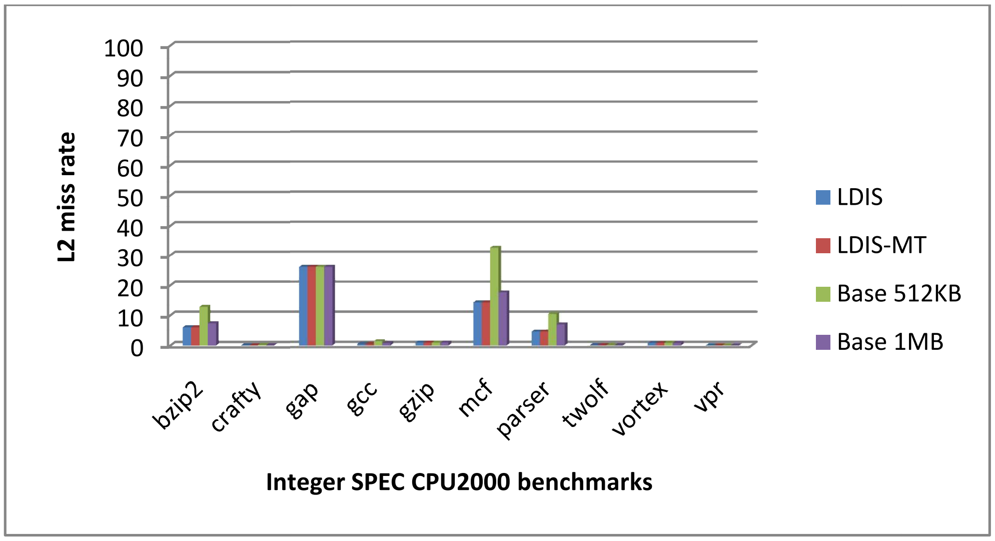

Fig 2: Variation of L2 miss rate with L2 cache configuration in Integer point benchmarks

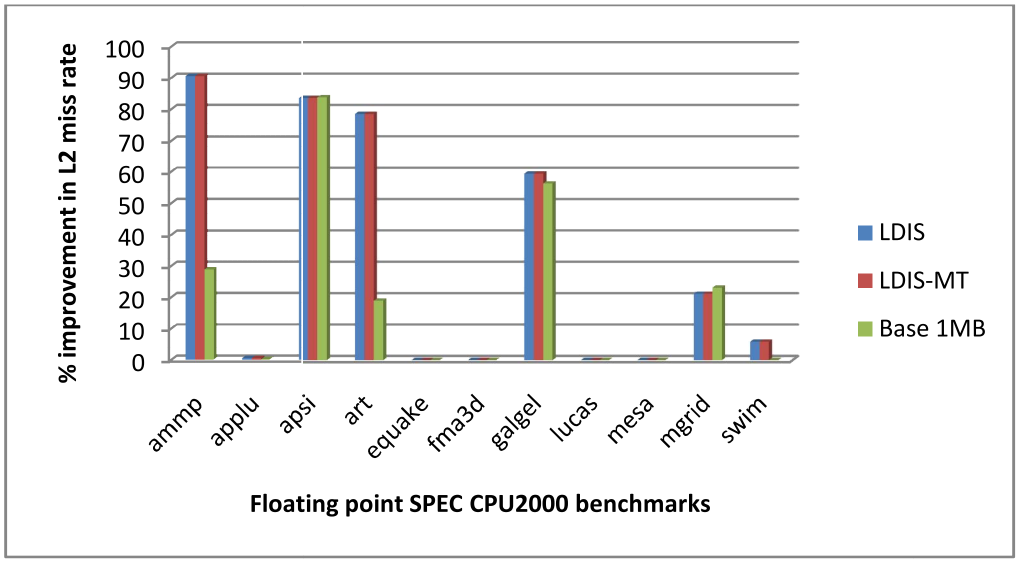

Fig 3: % improvement of L2 miss rate with L2 cache configuration in Floating point benchmarks

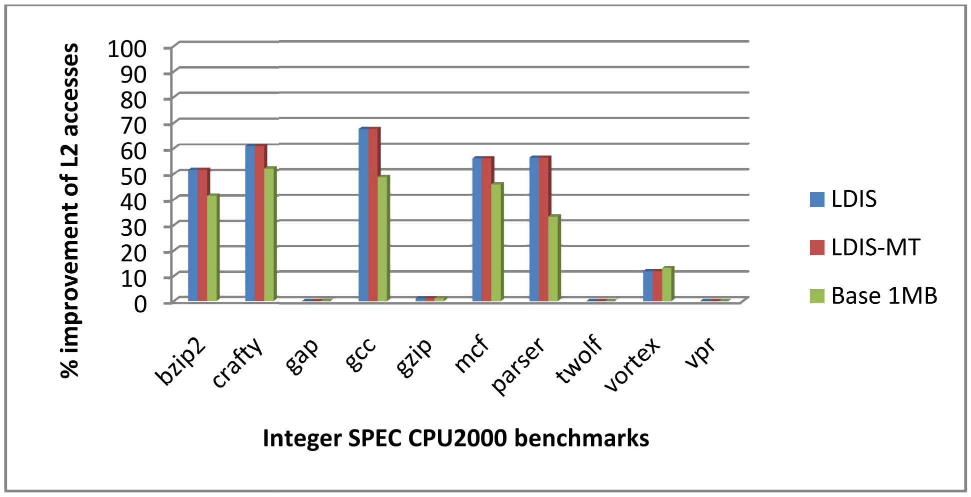

Fig 4: % improvement of L2 miss rate with L2 cache configuration in Floating point benchmarks

It can be observed from above figures that line distilled caches can achieve significantly high cache performance by decreasing L2 miss rate in floating point and integer benchmarks. It can also be observed that few benchmarks do not show any improvement. This is because the memory accesses by these benchmarks cause very less conflict misses in L2 cache. The LDIS achieves significant cache improvement in benchmarks where L2 cache conflicts are more.

It can also be observed that line distilled cache performs better than cache twice its size in most of the benchmarks. This is because LDIS exploits spatial locality more aggressively than cache double its size. In benchmarks where spatial locality is low bigger caches will outperform smaller LDIS caches.

5.2 HIT-MISS Distribution:

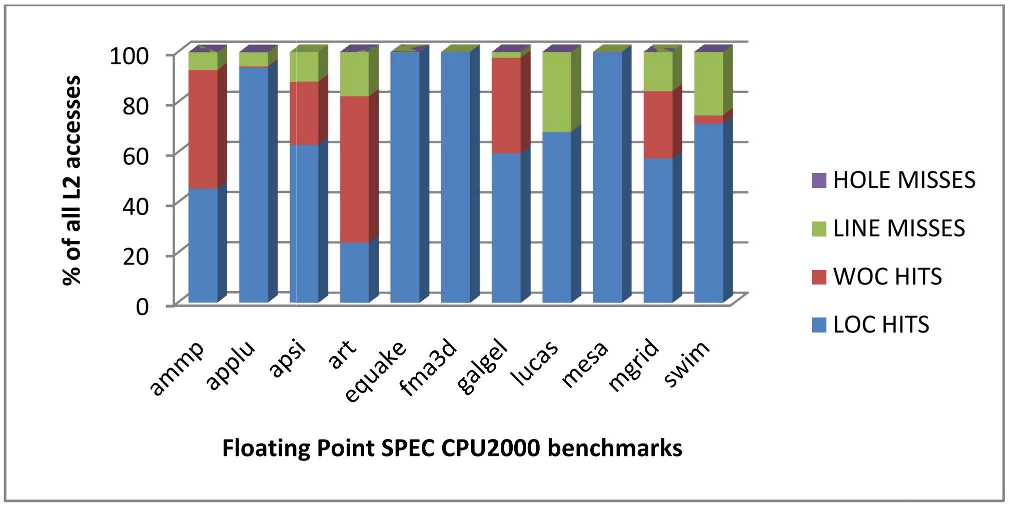

Fig 5: % Breakdown of cache access in Line Distilled L2 Cache in Floating point benchmarks

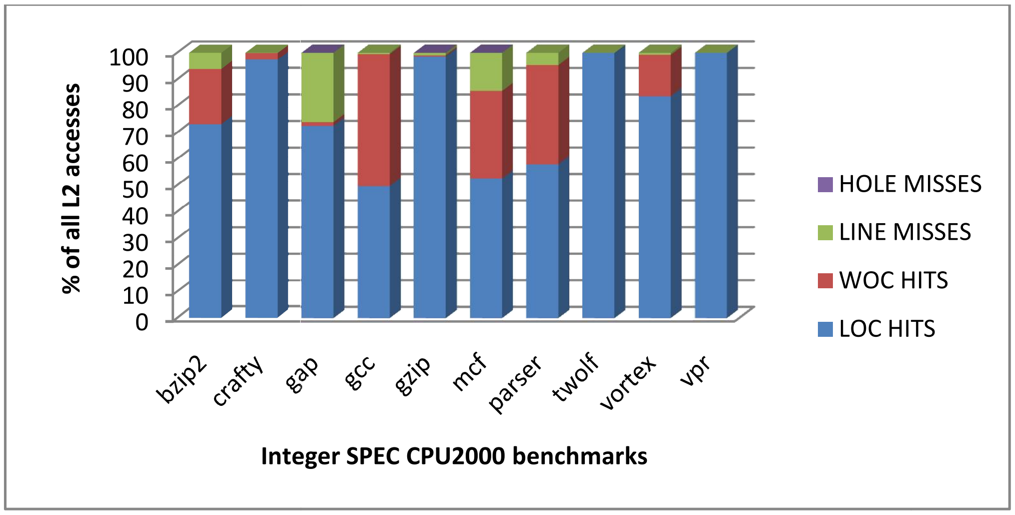

Fig 6: % Breakdown of cache access in Line Distilled L2 Cache in Integer benchmarks

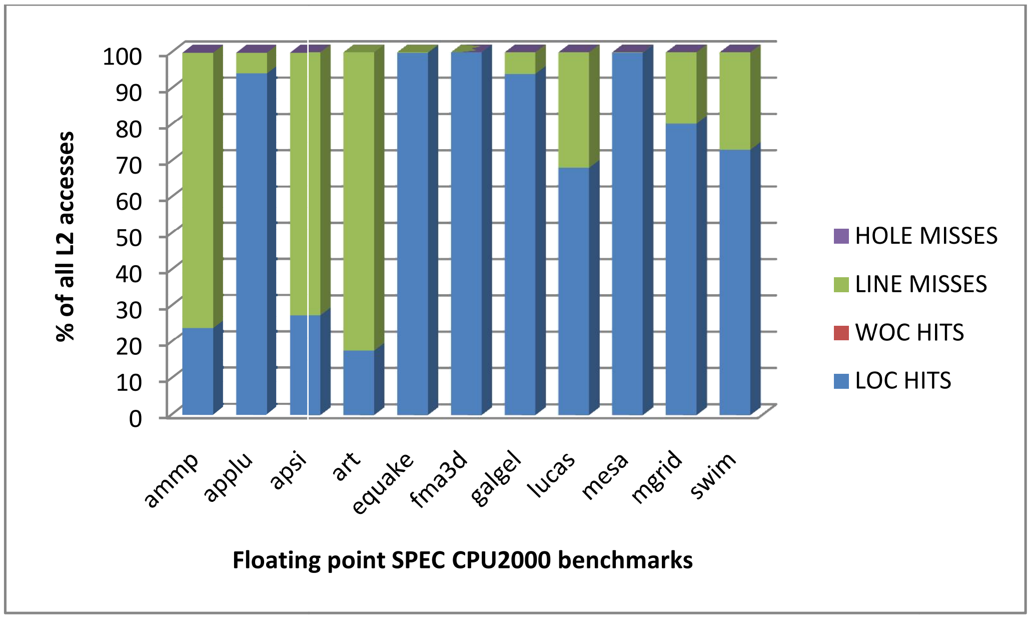

Fig 6: % Breakdown of cache access in 512KB L2 Cache in Floating Point benchmarks

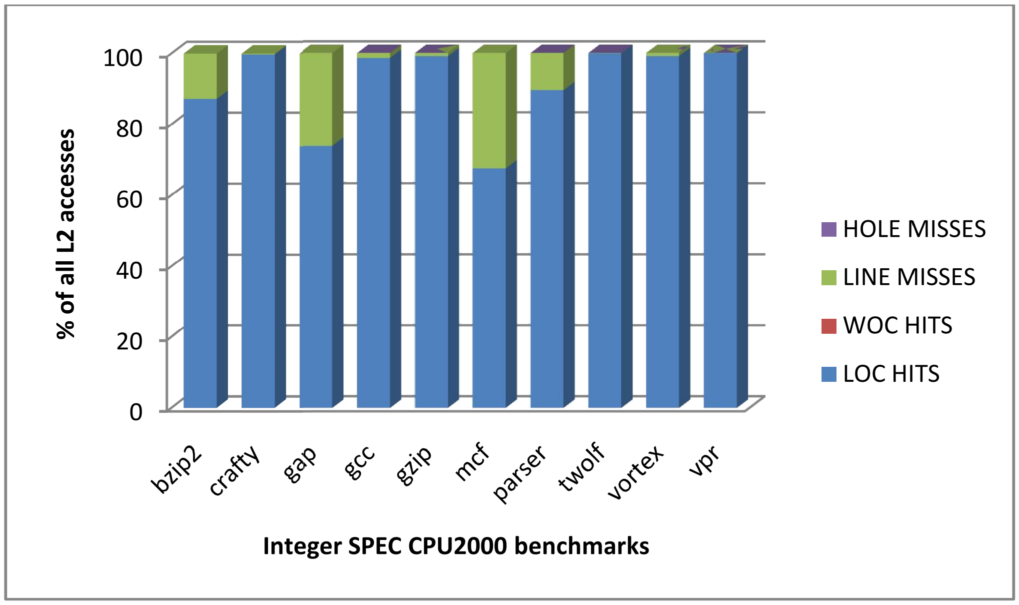

Fig 7: % Breakdown of cache access in 512KB L2 Cache in Integer benchmarks

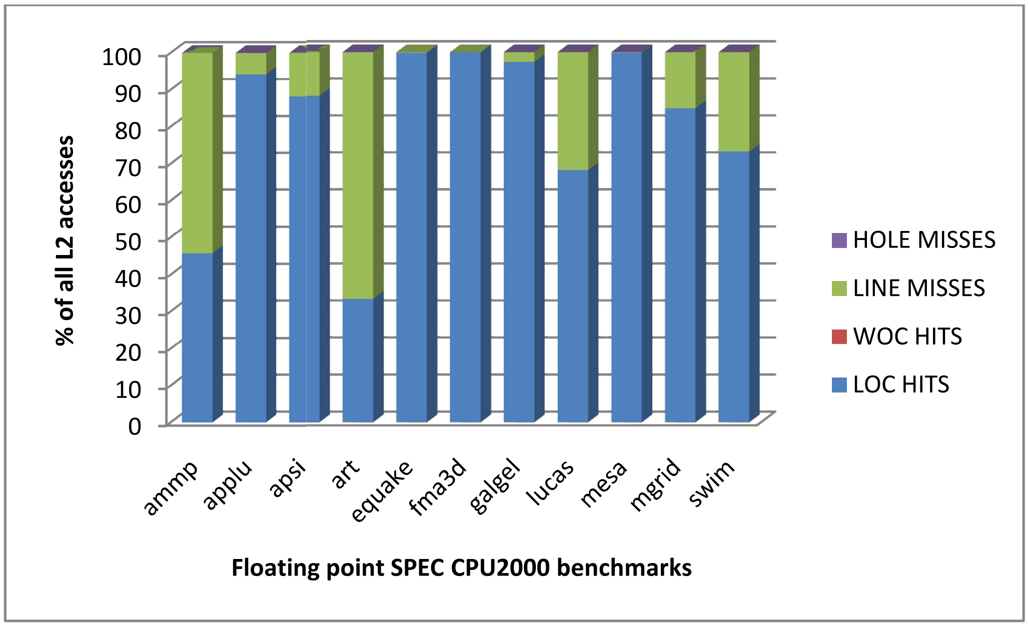

Fig 8: % Breakdown of cache access in 1MB L2 Cache in Floating Point benchmarks

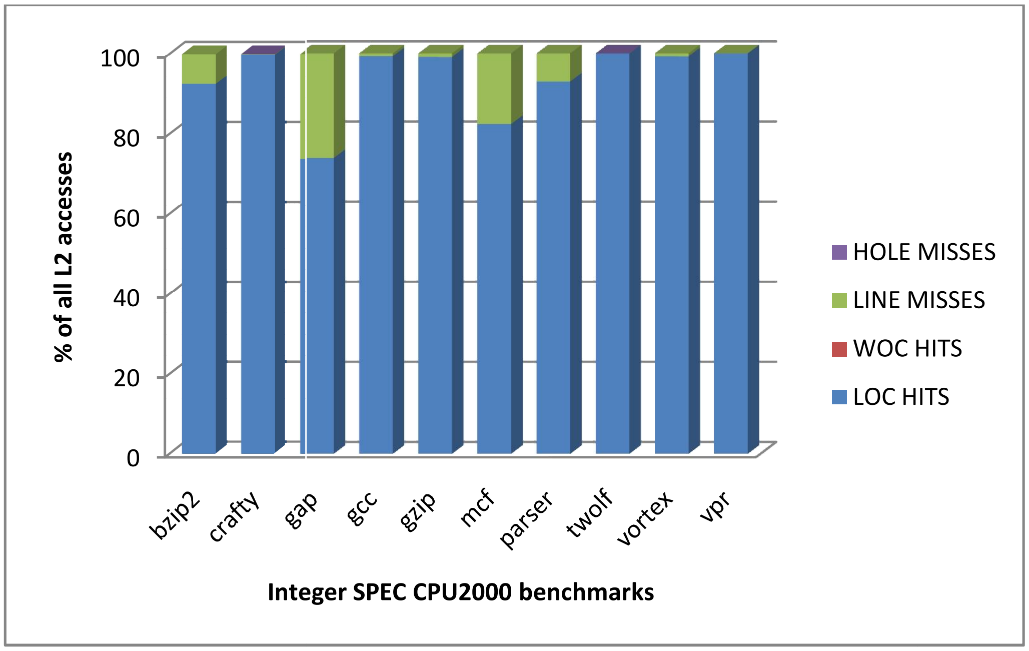

Fig 9: % Breakdown of cache access in 1MB L2 Cache in Integer benchmarks

It can be observed that in few benchmarks, large fraction of WOC hits help distill cache outperform baseline cache and cache double its size. In few benchmarks LOC hits are greater than hits in baseline cache, this is because of datasets larger than size of cache causes thrashing with LRU employed in baseline cache. Distilled cache reduces thrashing and hence show increased number of hits.

5.3 Overheads of Distillation:

Storage:

Size of each tag-entry in WOC (valid + dirty + head-bit + 23-bit tag + 3-bit word-id)	29 bits
Total number of tag-entries in WOC (1k sets * 2ways/set * 8entries/way)	16k
Overhead of tag-entries in WOC (29 bits/entry * 16k entries)	58kB
Total number of tag-entries in LOC(512KB/64B)	8k
Overhead of footprint bits in LOC (8bits/line * 8k lines)	8kB
Total storage overhead of distill-cache (58kB+8kB+256B)	66 kB
Area of baseline L2 cache (64kB tags + 512KB data)	544 kB
% increase in L2 area with distill-cache (66kB/544kB)	12%

The distill-cache incurs a total storage overhead of 12% of the area of the baseline cache.

Latency:
In distill cache has more latency because of word level access in WOC. The word-level access and word-tag matching incur additional latency compared to normal caches.

6. Conclusion:

Line distilled caches improve cache performance by filtering out unused words of least recently used cache lines and using the extra storage created by it to store used words of multiple evicted cache lines. Line distilled caches achieves lower miss rate by exploiting spatial locality aggressively.

However the improved performance comes at an area overhead (~12% for 512KB/64B cache, overhead % decreases as cache size increases) and slightly increased hit latency. Hence, line distill caches are not an attractive solution to improve cache performance at L1 level because of increased latency and area overhead. However, they are attractive solutions to improve cache performance at lower levels like L2 and L3 where caches are bigger and variation in hit-latency does not affect cache performance.

7. Author's Note:

Kindly note that our performance evaluations will be different than evaluations shown in [1]. This is because; for accurate results level 1 caches have to be sector caches as mentioned in [1]. Since simplescalar simulator employs only conventional cache model, our results will vary from the results in [1]. However the expected observations and inferences are the same.

8. References

[1] M. K. Qureshi, M. A Suleman, Yale N. Patt Line Distillation: Increasing Cache Capacity by filtering unused words in cache lines. HPCA-2007, pages 250-2007