Miss Caches, Victim Caches and Stream Buffers

Three optimizations to improve the performance of L1 caches

Paul Wicks

What's the Problem?

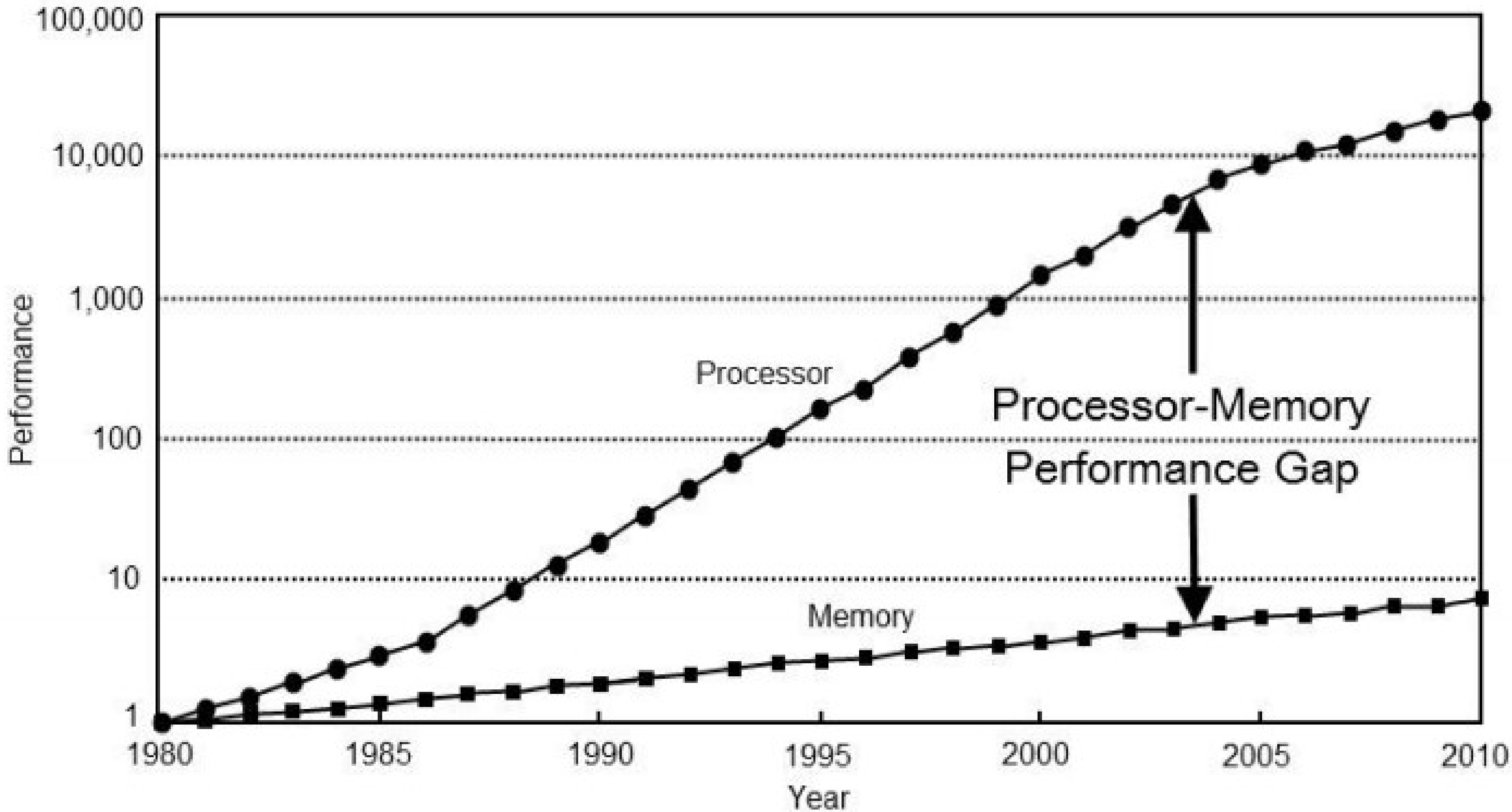


Figure 5.2 from *"Computer Architecture: A Quantitative Approach", Hennesy and Patterson*

Base System for Testing

- 1000 MIPS CPU
- L1 cache
  - 4KB data /4KB instr.
  - Direct-mapped
  - Miss Cost: 24 instr.
- L2 cache is
  - 1MB
  - Miss Cost: 320 instr.

- All measurements are against a selection of small benchmarks

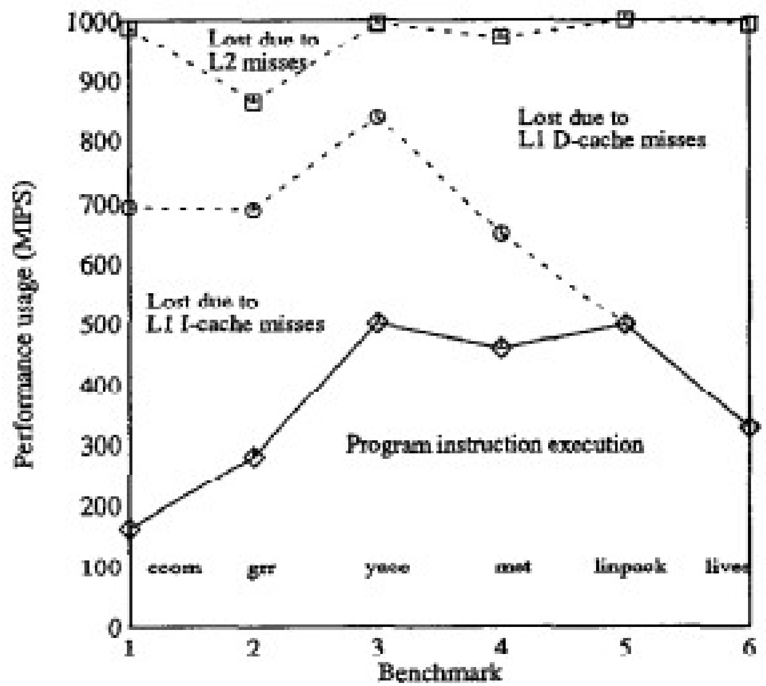


Figure 2-2 from *Norman P. Jouppi. 1990. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. SIGARCH Comput. Archit. News 18, 3a*

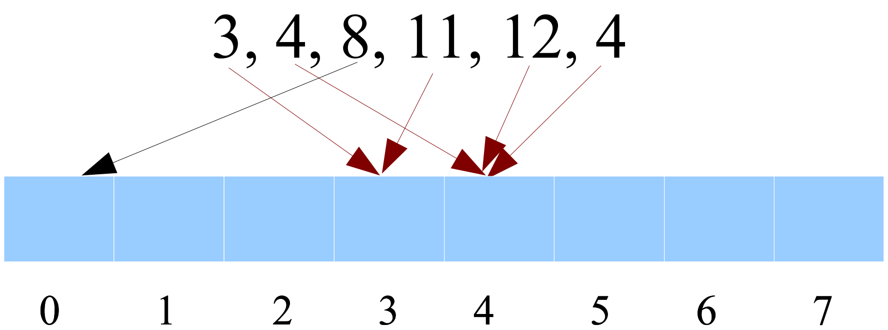
Basic Solution

- Key Insight of paper: Exploit locality of data and instruction streams to reduce cache misses
- Jouppi proposes 3 optimizations to do this
    - Miss Cache
    - Victim Cache
    - Stream Buffer

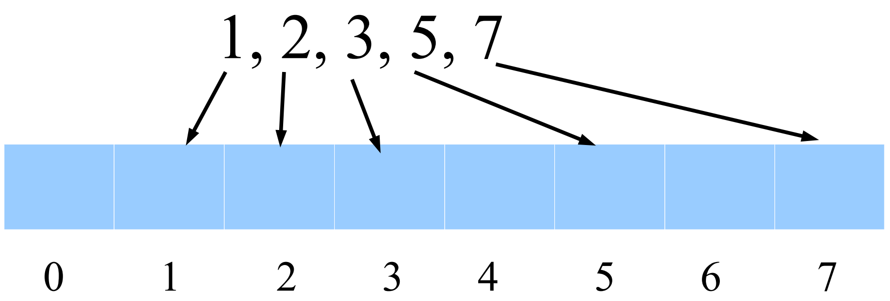
Classifying Cache Misses

Or, the 4 Cs

- Conflict



- Compulsory



- Capacity
  - Occurs when cache is not large enough to contain needed blocks

- Coherence
  - Occurs when an invalidate is issued by another processor in a multi-processor system

What is Locality?

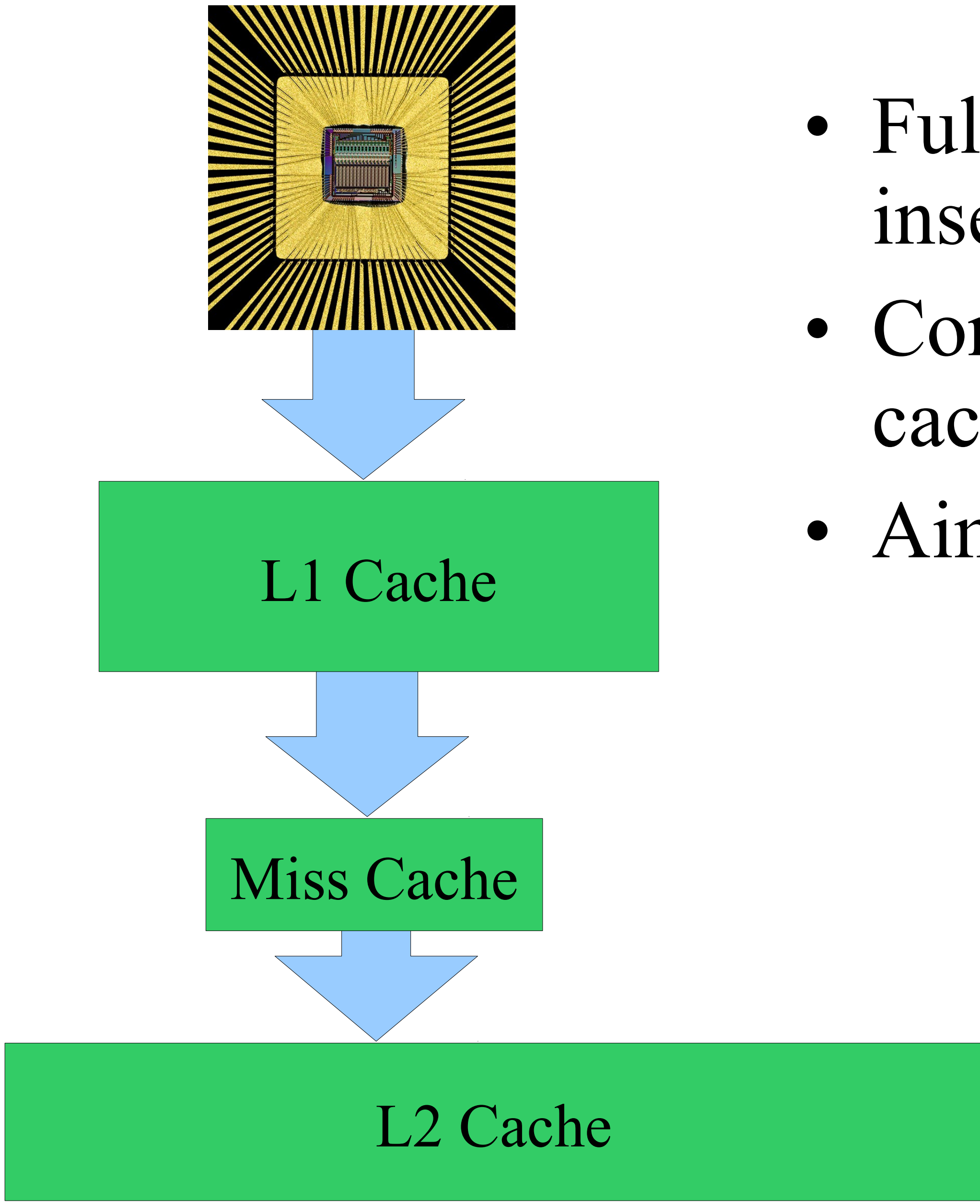
Temporal Locality: data used close to other data in time

3, 207,4, 3, 54, 207, 3

Spacial Locality: data close to other data in space (instructions often have this (at least until a function gets called...))

8, 23, 77, 78, 79, 80, 105

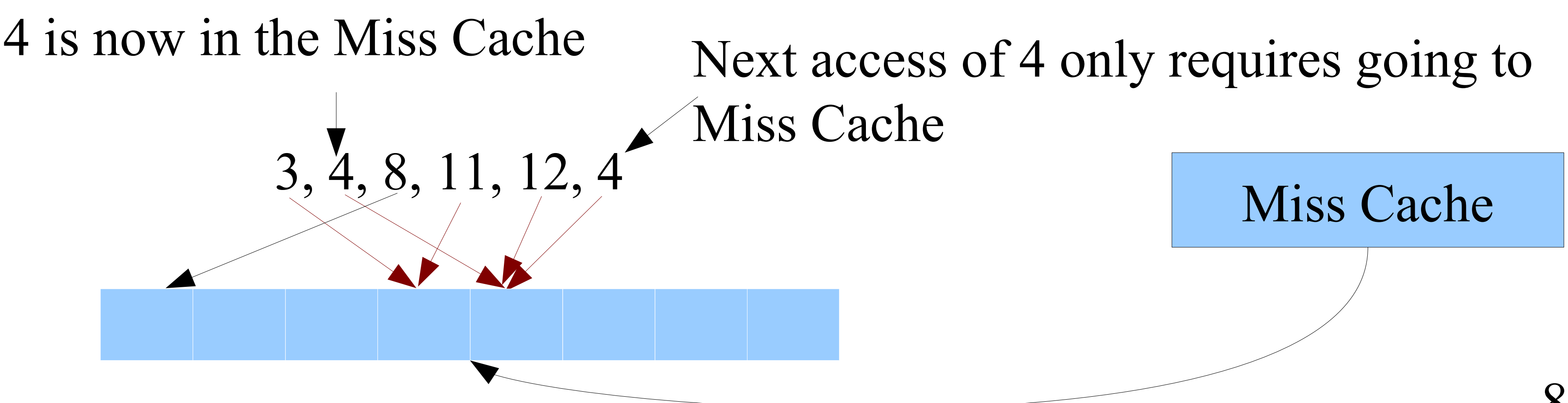
Optimization #1: The Miss Cache



- Fully-associative cache inserted between L1 and L2
- Contains between 2 and 5 cache lines of data
- Aims to reduce conflict misses

Miss Cache Operation

- On a miss in L1, we check the Miss Cache.
- If the block is there, then we bring it into L1
  - So the penalty of a miss in L1 is just a few cycles, possibly as few as one
- Otherwise, fetch the block from the lower-levels, but store the retrieved value in the Miss Cache



Miss Cache Performance

Cache Size	% Data Misses Removed	% Total Misses Removed
2 Entry Cache	25%	13%
4 Entry Cache	36%	18%

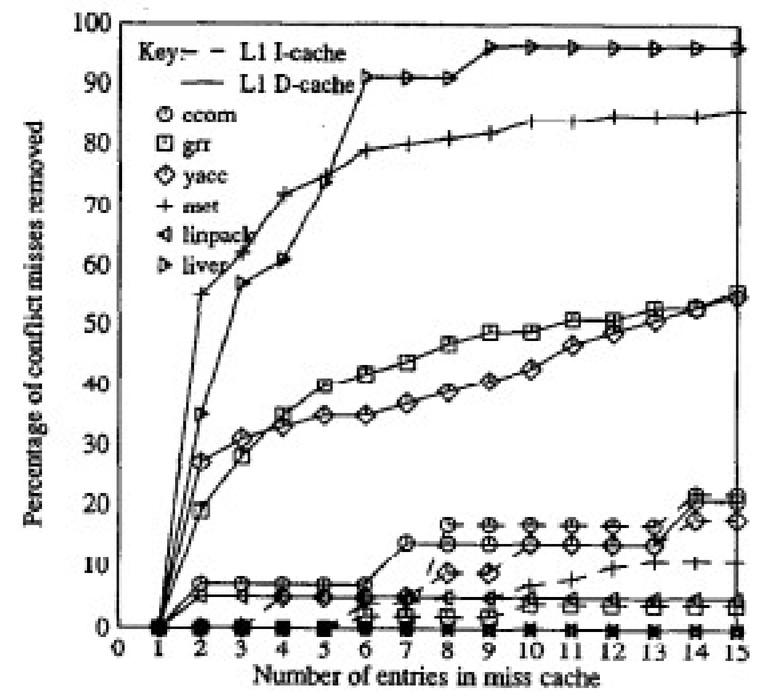
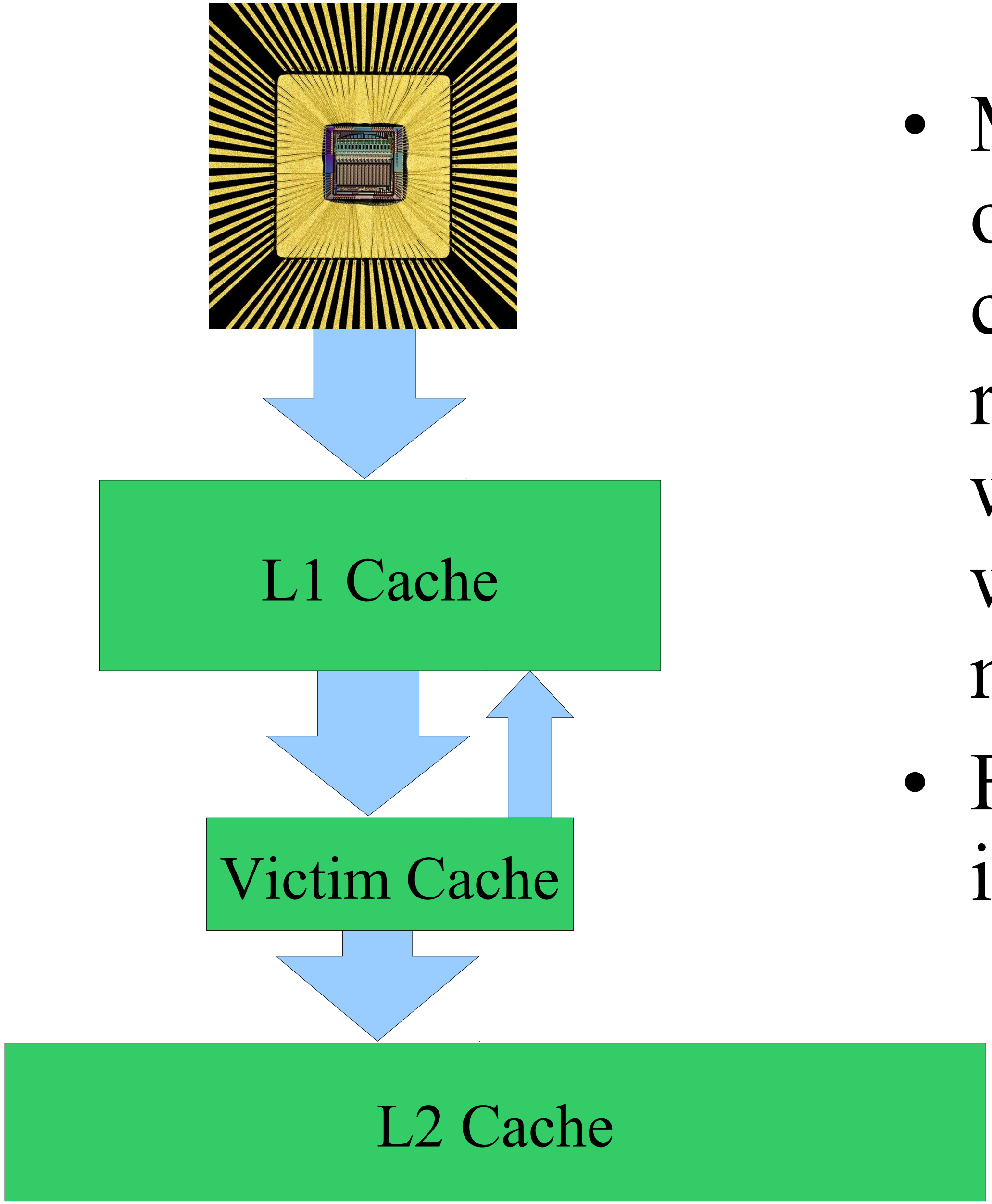


Figure 3-3: Conflict misses removed by miss caching

- What if we doubled the size of the L1 cache instead?
  - 32% decrease in cache misses
  - So only .13% decrease per line
- However, note that we are storing every block in the Miss Cache twice...

Figure 3-3 from *Norman P. Jouppi. 1990. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. SIGARCH Comput. Archit. News 18, 3a*

Optimization #2: The Victim Cache



- Motivation: Can we improve on our miss rates with miss caches by modifying the replacement policy (i.e. can we do something about the wasted space in the pure miss caching system)?
- Fully-associative cache inserted between L1 and L2

Victim Cache Operation

- On a miss in L1, we check the Victim Cache
- If the block is there, then bring it into L1 and swap the ejected value into the miss cache
  - Misses that are caught by the cache are still cheap, but better utilization of space is made
- Otherwise, fetch the block from the lower-levels

Victim Cache Performance

- Even better than Miss Cache!

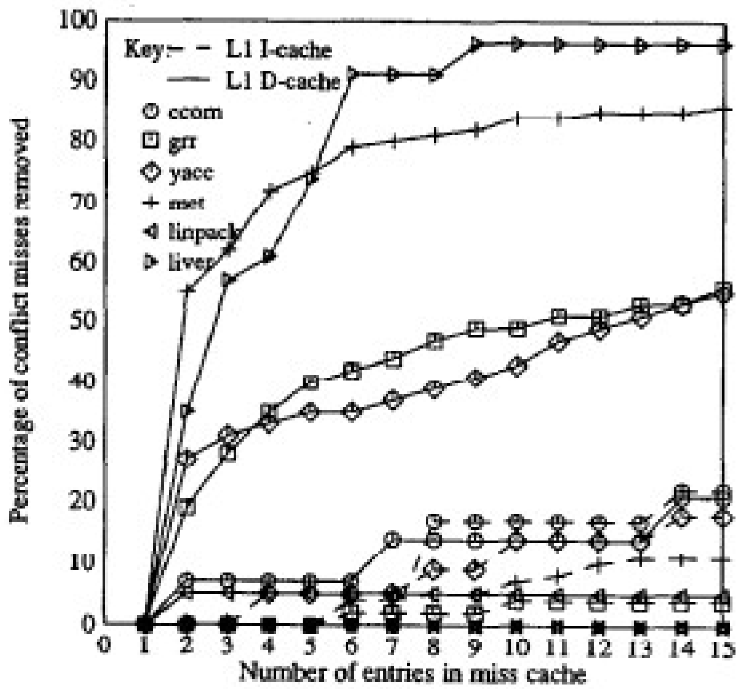


Figure 3-3: Conflict misses removed by miss caching

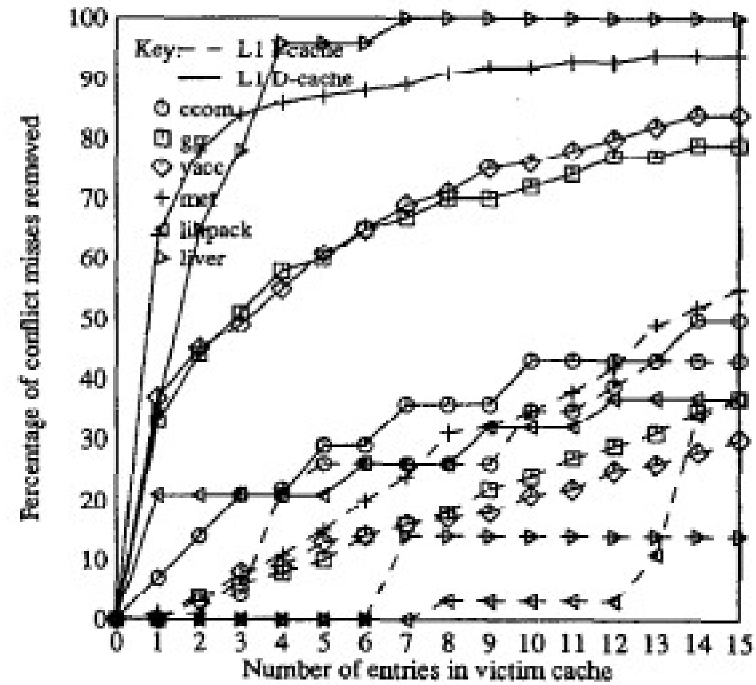


Figure 3-5: Conflict misses removed by victim caching

- Smaller L1 caches benefit more from victim caches

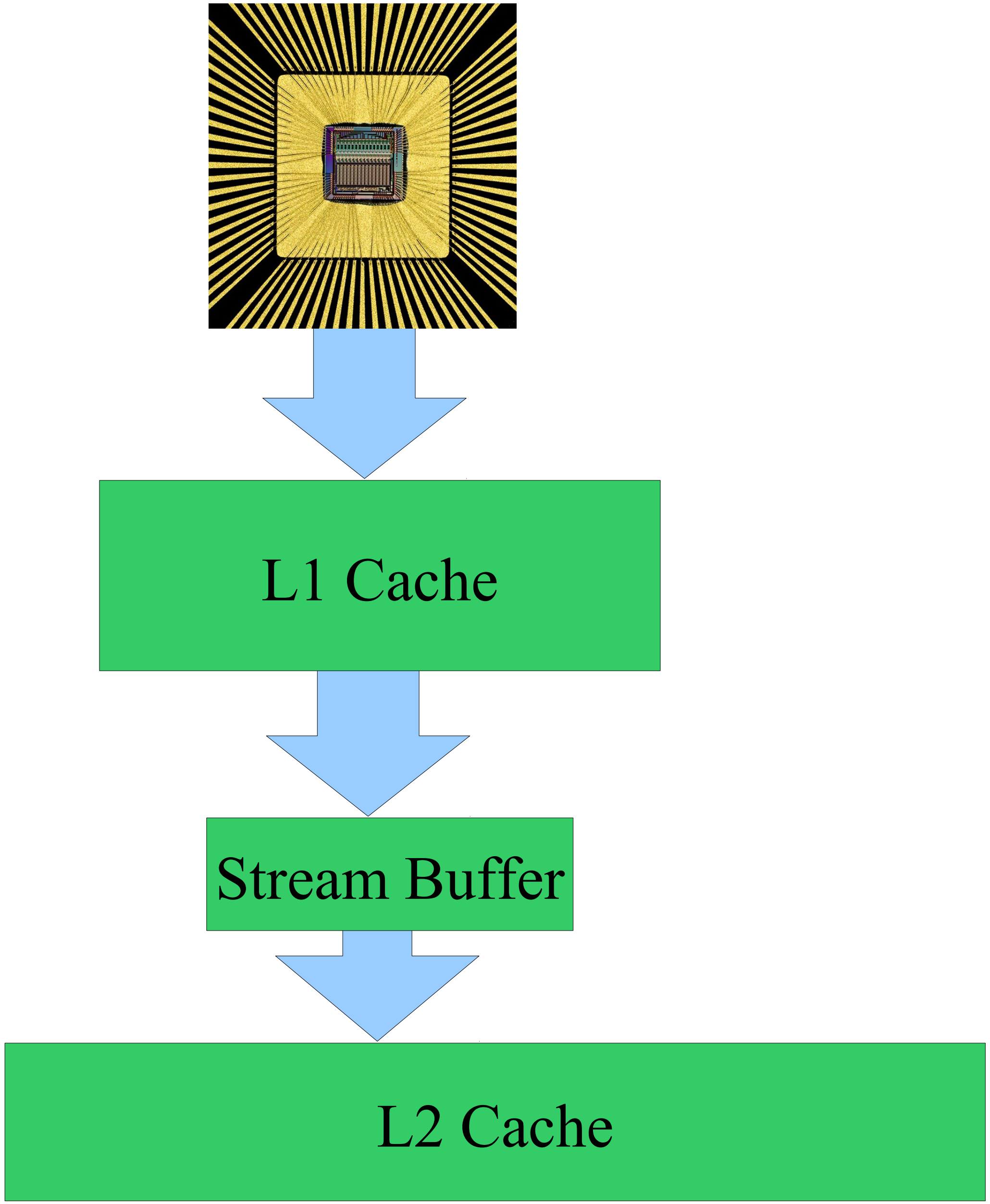
Figures 3-3,3-5 from *Norman P. Jouppi. 1990. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. SIGARCH Comput. Archit. News 18, 3a*

Wait a minute!

- What about compulsory and capacity misses?
- What about instruction misses?
- Victim Cache and Miss Cache most helpful when temporal locality can be exploited
- Prefetching techniques can help
  - Prefetch Always
  - Prefetch on Miss
  - Tagged Prefetch

- Can we improve on these techniques?

Optimization #3: Stream Buffers



- Stream Buffer is a FIFO queue placed in between L1 and L2

How do they work

- When a miss occurs in L1, say at address A, the Stream Buffer immediately starts to prefetch elements at A+1
- Subsequent accesses check the head of the Stream Buffer before going to L2
- Note that non-sequential misses will cause the line to restart prefetching (i.e. A+2 and A+4 will each restart the prefetching process, even if A+4 was already in the stream)

Stream Buffer Performance

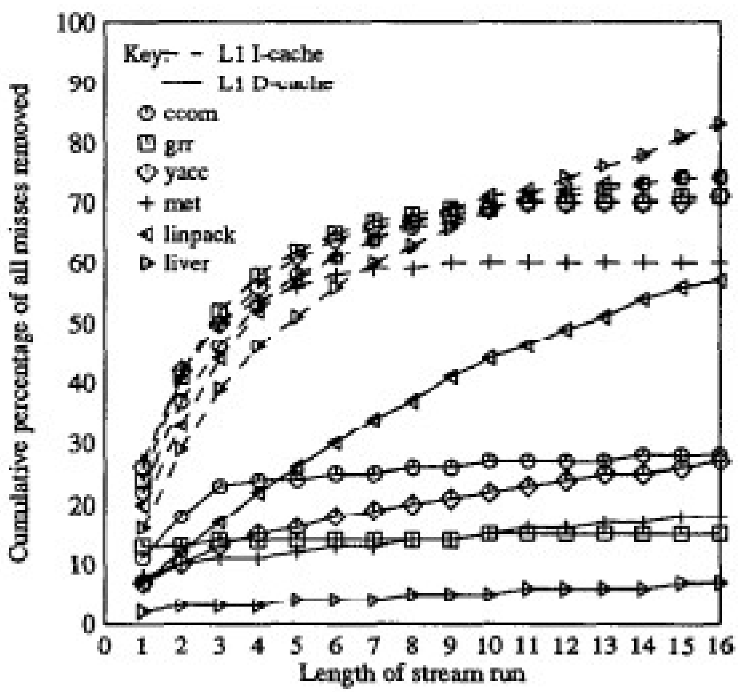


Figure 4-3: Sequential stream buffer performance

- 72% of instr. misses removed
- 25% of data misses removed

Figure 4-3 from *Norman P. Jouppi. 1990. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. SIGARCH Comput. Archit. News 18, 3a*

Further refinements

- What about situations where we are drawing from several streams simultaneously (e.g. reading from several rows of an array)?
- Jouppi proposes Multi-Way Stream Buffers
  - Several Stream Buffers in parallel
  - Performs well, especially on data streams (43% improvement vs. 25% improvement for basic streams)

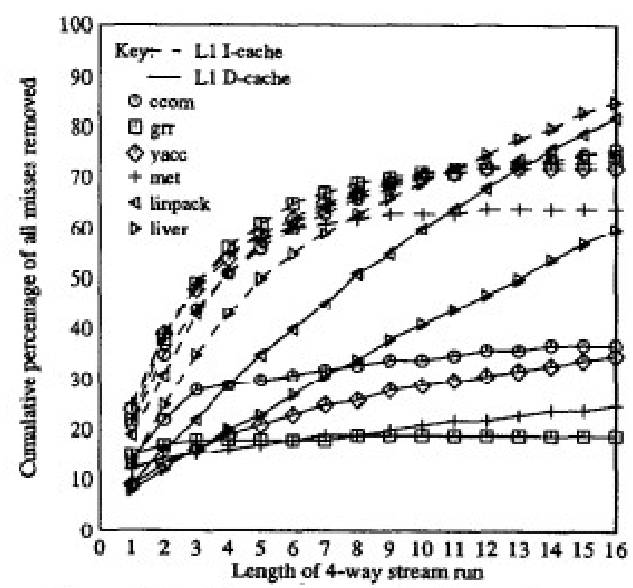


Figure 4-5: Four-way stream buffer performance

Figure 4-5 from *Norman P. Jouppi. 1990. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. SIGARCH Comput. Archit. News 18, 3a*

Summary of techniques

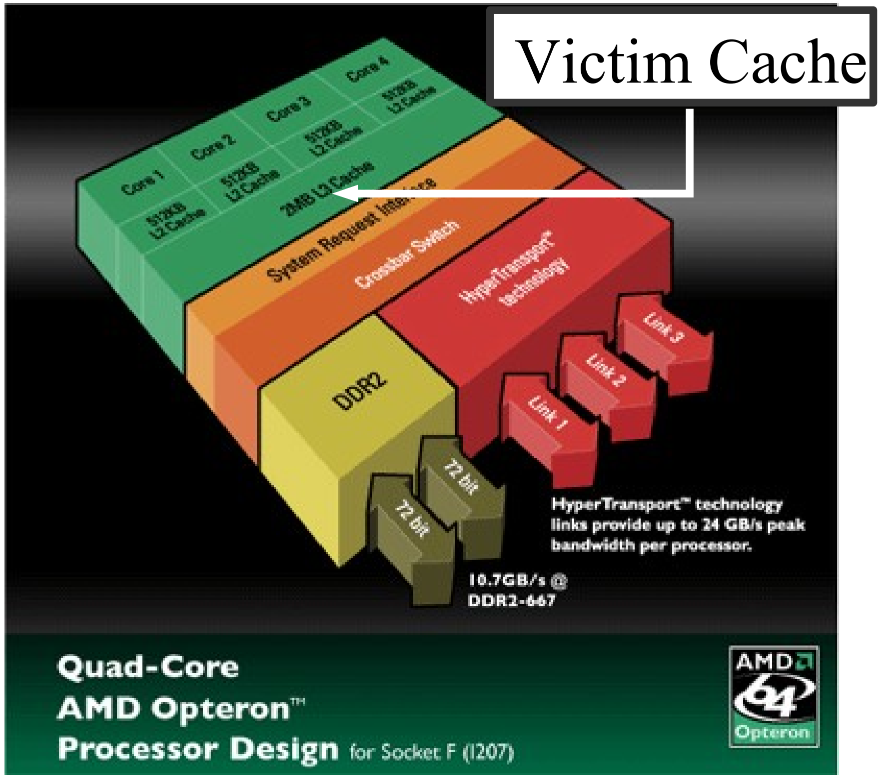
- Miss caches: Small caches in between L1 and L2. Whenever a miss occurs in L1, they receive a copy of the data from the lower level cache
- Victim cache: Enhancement of Miss Caches. Instead of copying data from lower levels on a miss, they take whatever block has been evicted from L1
- Stream Buffers: Simple FIFO buffers that are immediately prefetched into on a miss

What do other think of these ideas?

- Many extensions have been proposed to the optimizations presented in the Joupi paper
  - *S. Palacharla and R. E. Kessler. 1994. Evaluating stream buffers as a secondary cache replacement. SIGARCH Comput. Archit. News 22, 2 (April 1994), 24-33.*
    - Stream Buffer as L2 cache
    - Better detection of streams via history buffer
    - Non-unit length stride
  - *Dimitrios Stiliadis and Anujan Varma. 1997. Selective Victim Caching: A Method to Improve the Performance of Direct-Mapped Caches. IEEE Trans. Comput. 46, 5 (May 1997), 603-610.*
    - Prediction scheme to place incoming blocks less likely to be used in the victim cache
- 350+ papers cite this paper
- Won the 2005 International Symposium on Computer Architecture Influential Paper Award

Implementations

- Miss Caches and Stream Buffers used by HP
- Stream Buffers used by Cray
- Some recent AMD systems use a Victim Cache as a shared L3 cache, something which Jouppi speculates could be useful in the paper
- Others as well



How could this paper be improved?

- Some of the figures are confusing
- More data!
  - All the percentages are averages, is anything being hidden by those averages?
  - What about bigger programs? OS workloads?
  - To be fair, the paper admits that some of these things are problems
- Occasional lack of parallel structure: for example, it would be nice if we got the actual percentages for victim caches instead of just the graph

Conclusion

- Good paper overall: succinctly lays out problem and provides a clear avenue of attack

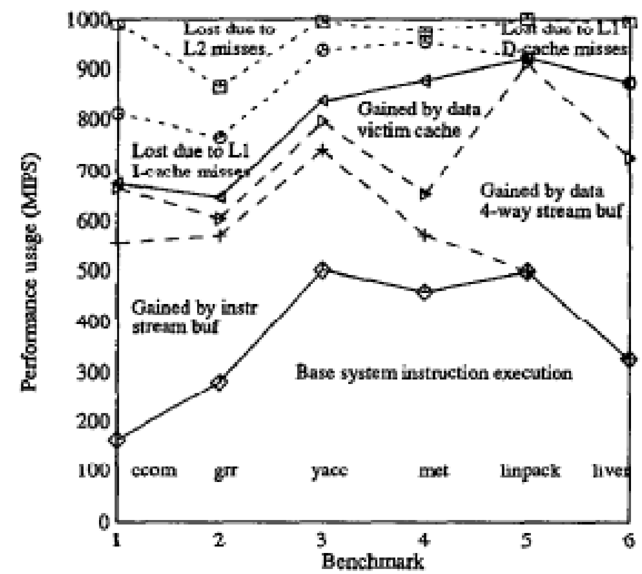


Figure 5-1: Improved system performance

- Paper is still relevant
  - See cite count
  - Problems that were bad in 1990 have not improved much

Thank you for your attention!

Questions?

Extra Figures

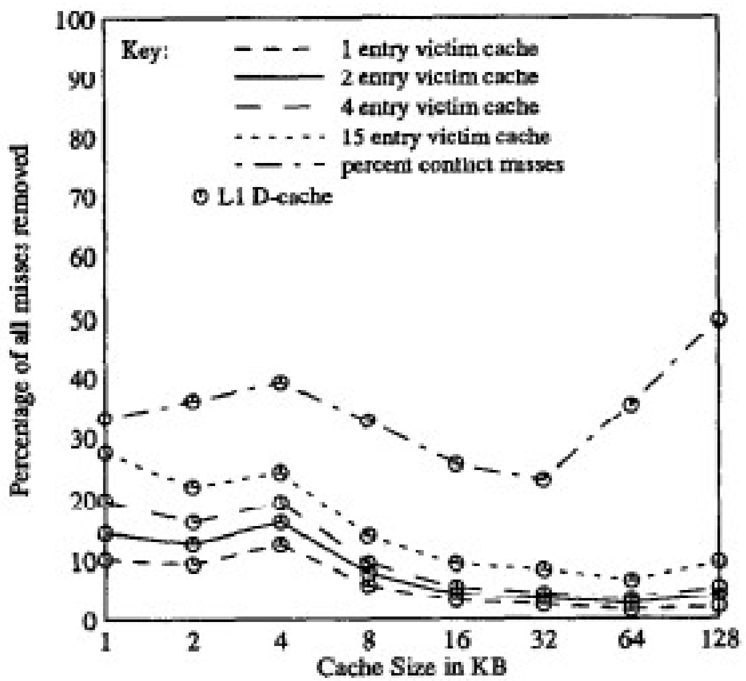


Figure 3-6: Victim cache: vary direct-map cache size

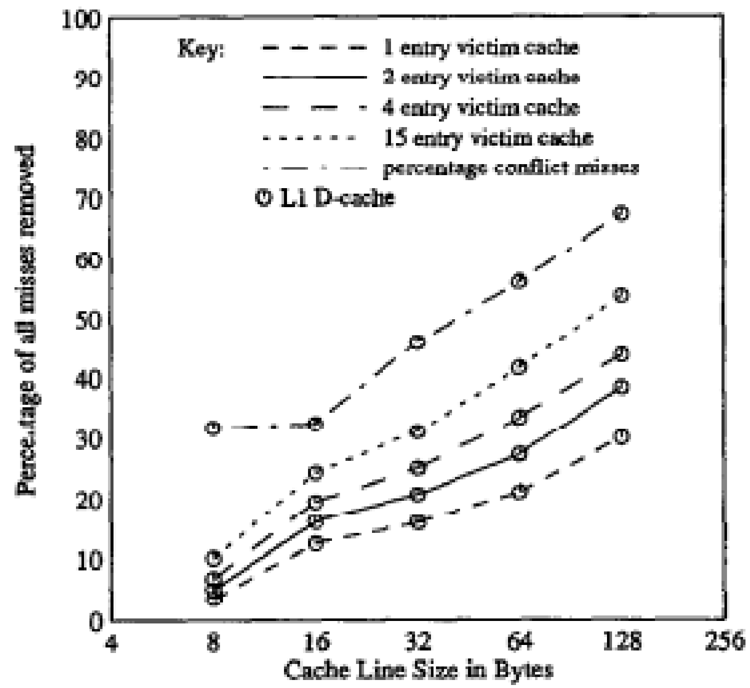


Figure 3-7: Victim cache: vary data cache line size