Cache Filtering Techniques to Reduce the Negative Impact of Useless Speculative Memory References on Processor Performance

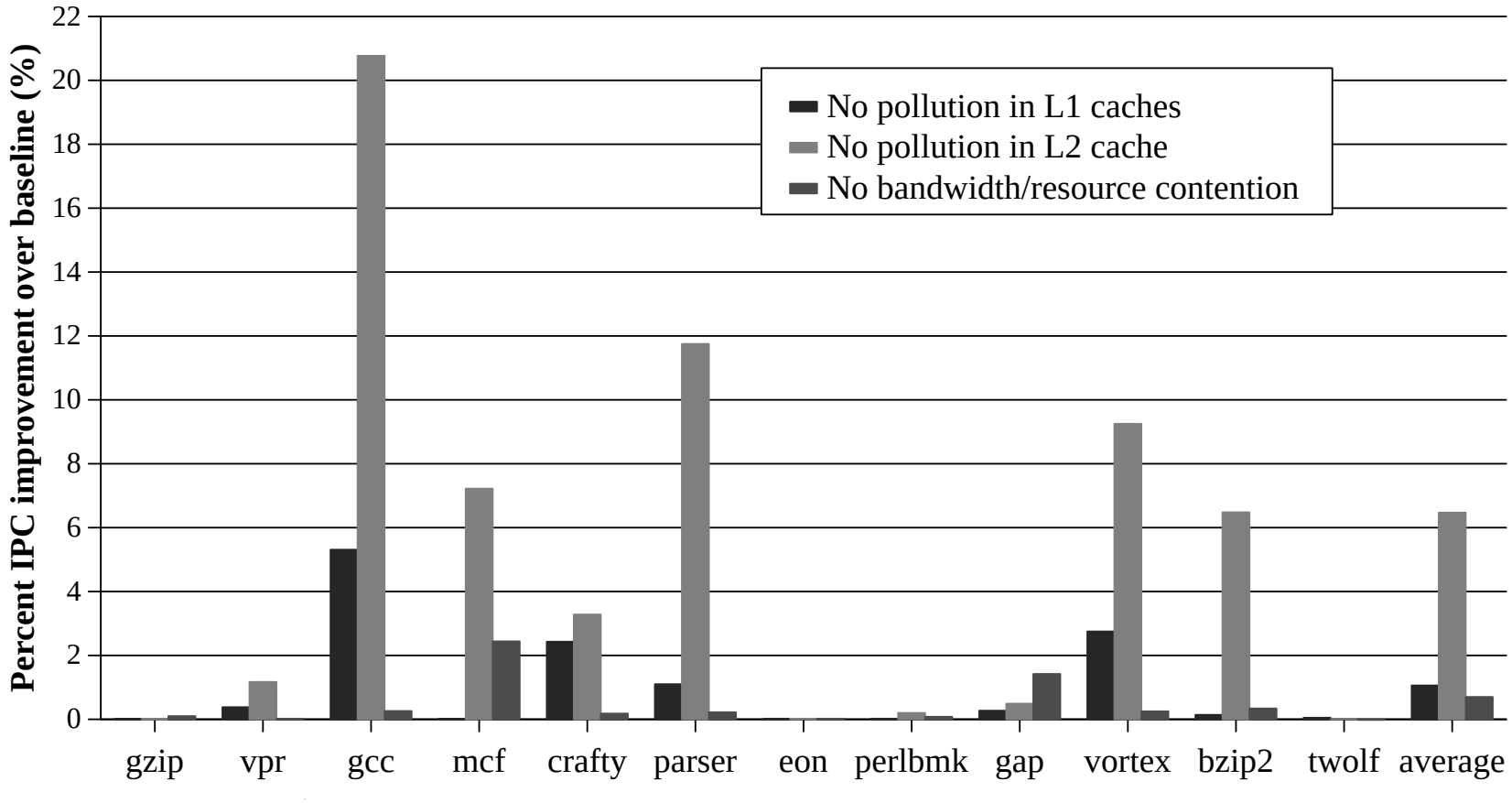
Onur Mutlu
Hyesoon Kim
David N. Armstrong
Yale N. Patt

The University of Texas at Austin

Motivation

- Branch prediction and prefetching are widely used by processors to improve performance.
- Incorrect branch predictions and inaccurate prefetch requests result in memory references that are not needed by correct execution: useless speculative memory references
- These useless references may be detrimental to processor performance because they cause
  - L1/L2 cache pollution
  - Bandwidth/resource contention

Why are Useless References Bad?

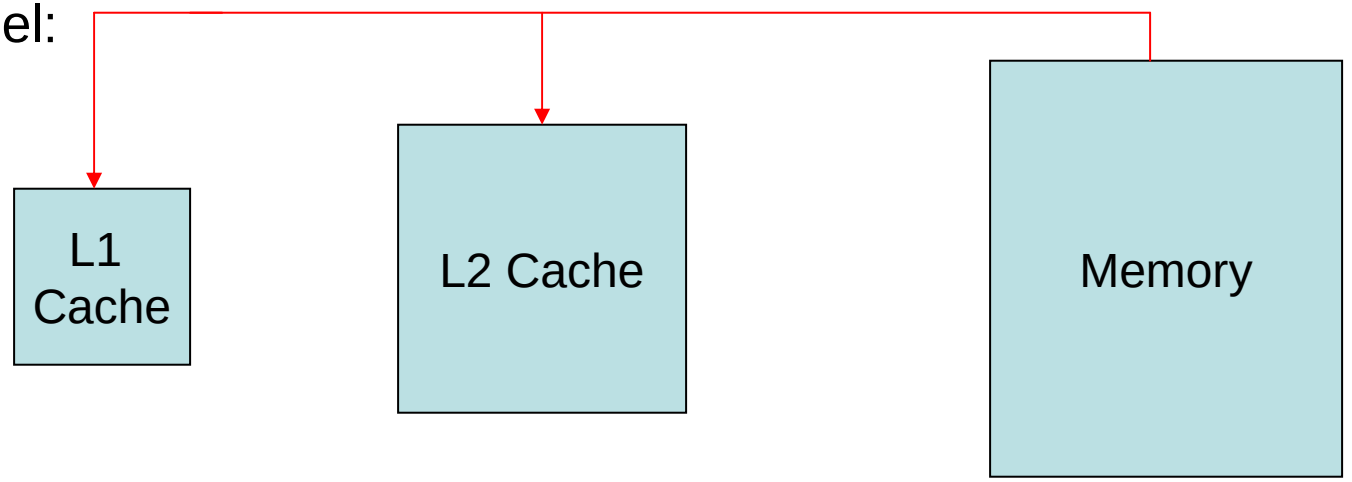


Talk Outline

- Motivation
  - Negative performance impact of speculative references is primarily due to L2 cache pollution
- Analysis of Speculative Memory References
- Solution (Cache Filtering Techniques)
- Experimental Evaluation
- Conclusion

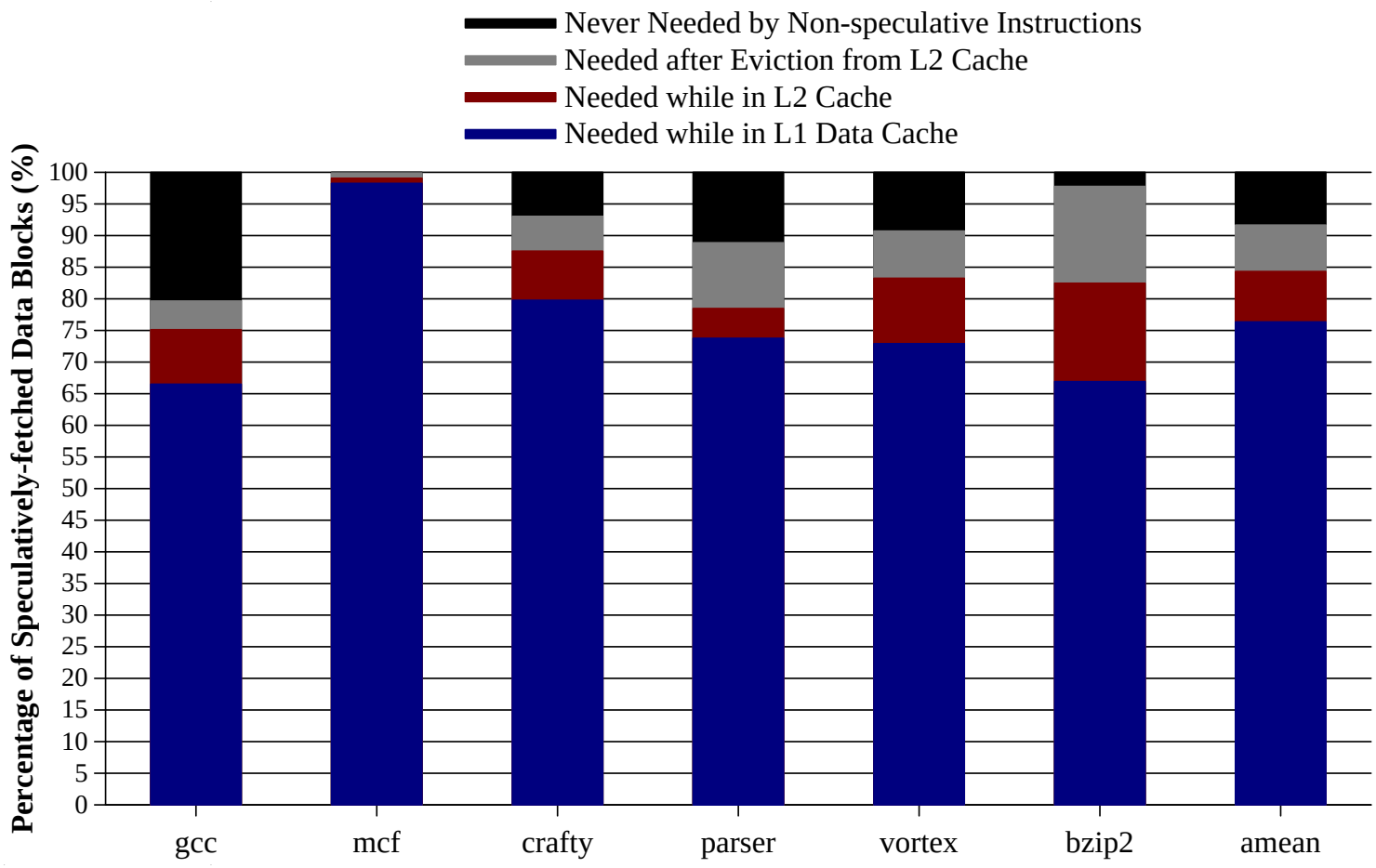
Speculative L2 Misses

- Current Model:



- A speculative L2-miss allocates a cache block in both L1 and L2 caches (like a non-speculative L2 miss)
- Useless speculative blocks occupy entries in both cache levels → pollution in both cache levels

Breakdown of Speculative Data Blocks



Two Observations on Speculative References

- Observation 1:

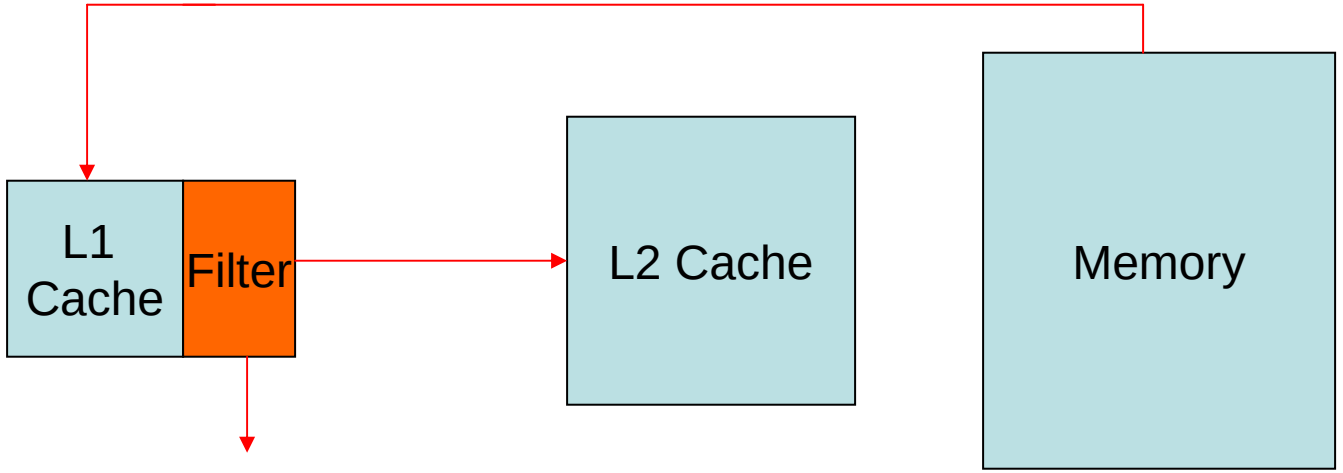
  If a speculatively-fetched cache block is needed by correct path execution, then it is most likely needed while it resides in the L1 cache.

- Observation 2:

  If a speculatively-fetched cache block is not needed while it resides in the L1 cache, then it is likely that the block will never be needed or it will be needed after it is evicted from the L2 cache.

Solution to L2 Pollution: L1 Cache as a Filter

- New Model:



- A speculative L2-miss allocates a cache block only in L1 (unlike a non-speculative L2 miss)
- A speculatively-fetched block is marked as *speculative* in the L1 cache
- If it is referenced by a non-speculative instruction while it is in the L1 cache, the block is written back into L2 when it is evicted.

L1 Cache as a Filter: Two Filtering Policies

- If a speculatively-fetched block is NOT referenced by a non-speculative instruction while it is in the L1 cache:
  - Filter the block out of the L2 cache to reduce L2 pollution

- Filtering Policy 1:

  It is NOT written back into L2 (no-spec-L2fill)
- Filtering Policy 2:

  It is written back into L2, but into the LRU slot of its set (spec-L2fill-lru)

Tradeoffs in Two Policies

- no-spec-L2fill policy:
  - + Eliminates all L2 pollution due to speculative references
  - - Filters out some useful speculatively-fetched blocks that would have been used if placed in L2.
- spec-L2fill-lru policy:
  - + Captures the benefit of useful speculatively-fetched blocks that are used shortly after being evicted from L1.
  - - Some L2 pollution due to speculative references remains, but the effect is less pronounced because a useless block occupies an L2 cache line for a shorter amount of time.

Implementation Cost

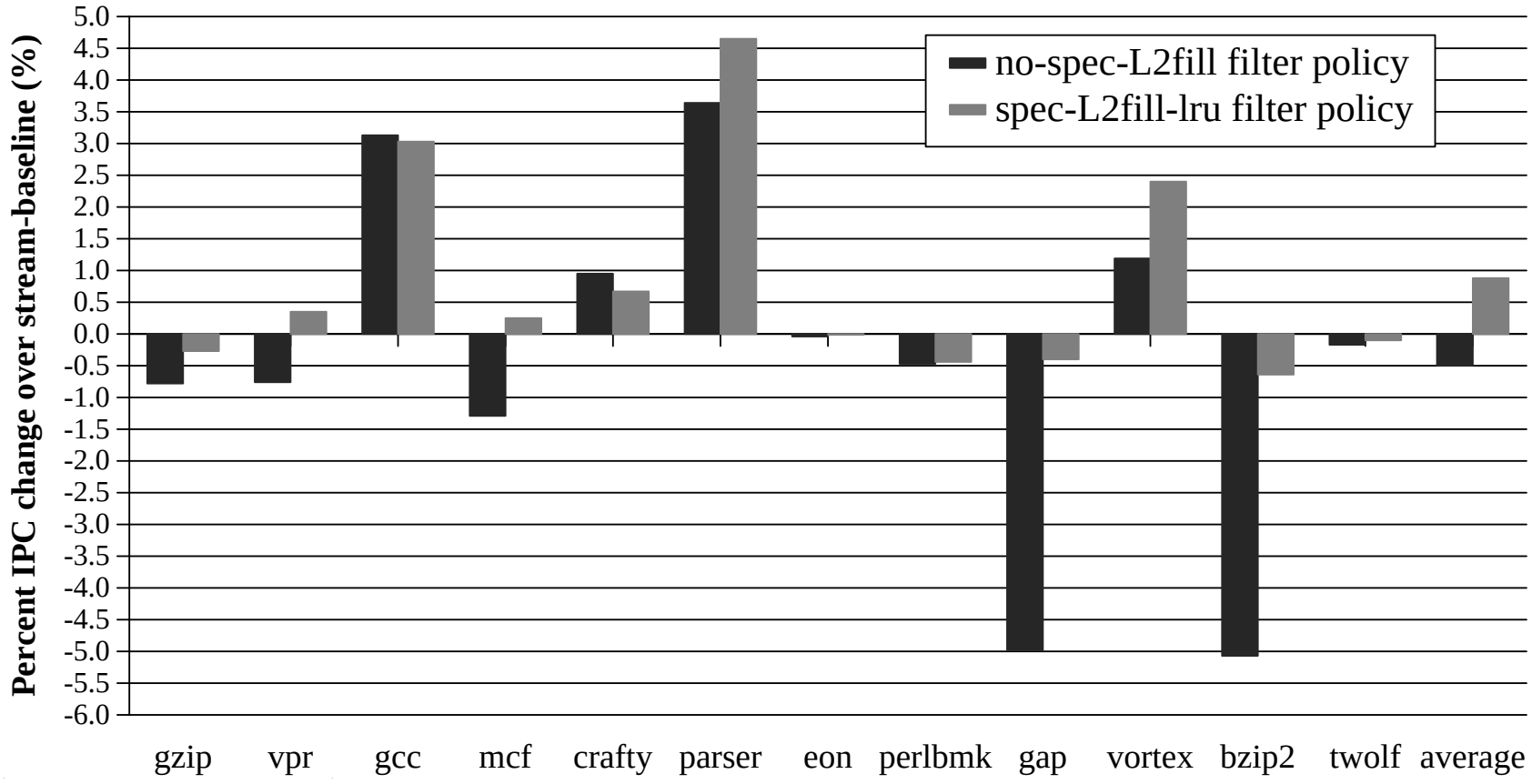
- The processor needs to distinguish between speculative and non-speculative L2 misses
  - Requires 1 bit per L2 miss buffer (MSHR) entry
  - All hardware prefetches are initially speculative
  - Instruction and data L2 miss requests are considered to be non-speculative until they are known to be speculative
    - On the resolution of a mispredicted branch all younger miss buffer entries are marked as speculative
- Each L1 cache block has an associated *speculative* bit
  - Set if the L2 miss was marked speculative in miss buffer
  - Reset if an instruction that accessed the cache block is retired
- Each L1 cache block has an associated *write-back* bit
  - Set if a speculative block is referenced by a non-speculative instruction

Experimental Evaluation

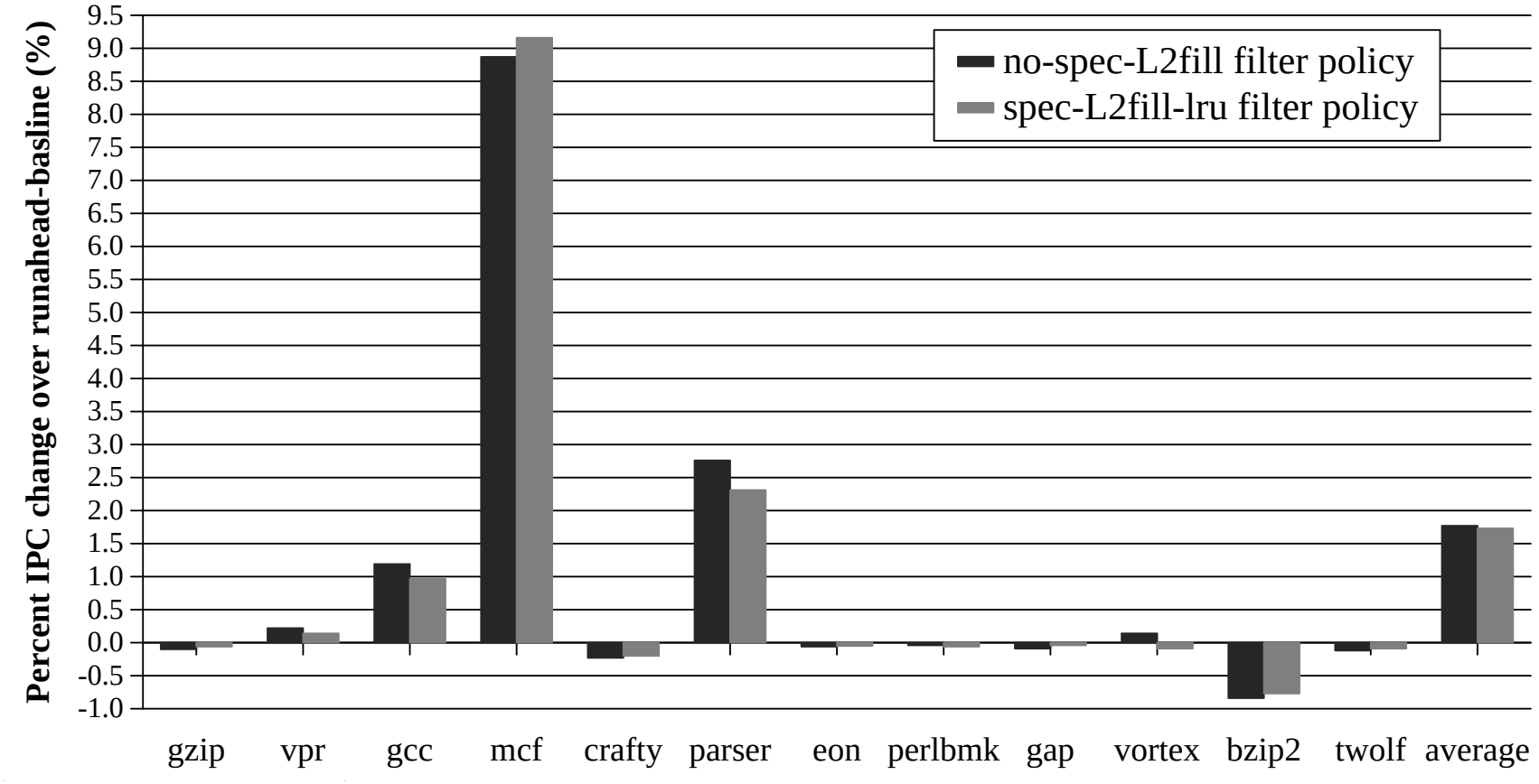
- 8-wide aggressive superscalar, out-of-order baseline, 128-entry instruction window
- Aggressive branch prediction
  - 64K-entry gshare, 64K-entry PAs hybrid, 64K-entry selector
- 64 KB, 4-way L1 Instruction and Data Caches
- 512 MB, 8-way Unified L2 cache
- Minimum 500-cycle main memory latency

- Evaluated the filtering mechanisms on two baselines:
  - *Stream-baseline*: with an aggressive hardware stream prefetcher
  - *Runahead-baseline*: with runahead execution [Mutlu et. al., HPCA'03], a method of aggressive speculative pre-execution under an L2 cache miss

IPC Delta of Filtering on *Stream-baseline*



IPC Delta of Filtering on *Runahead-baseline*

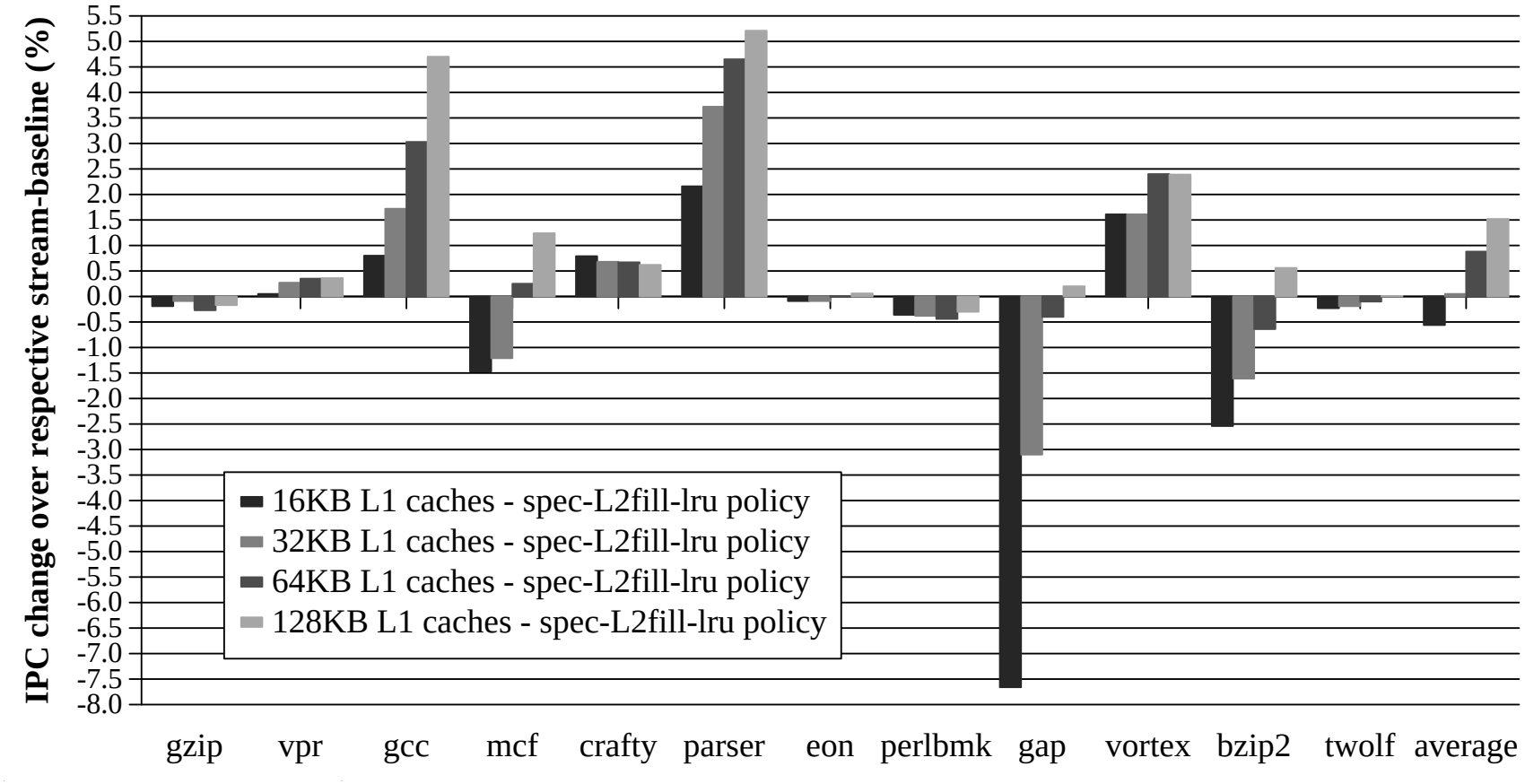


Observations

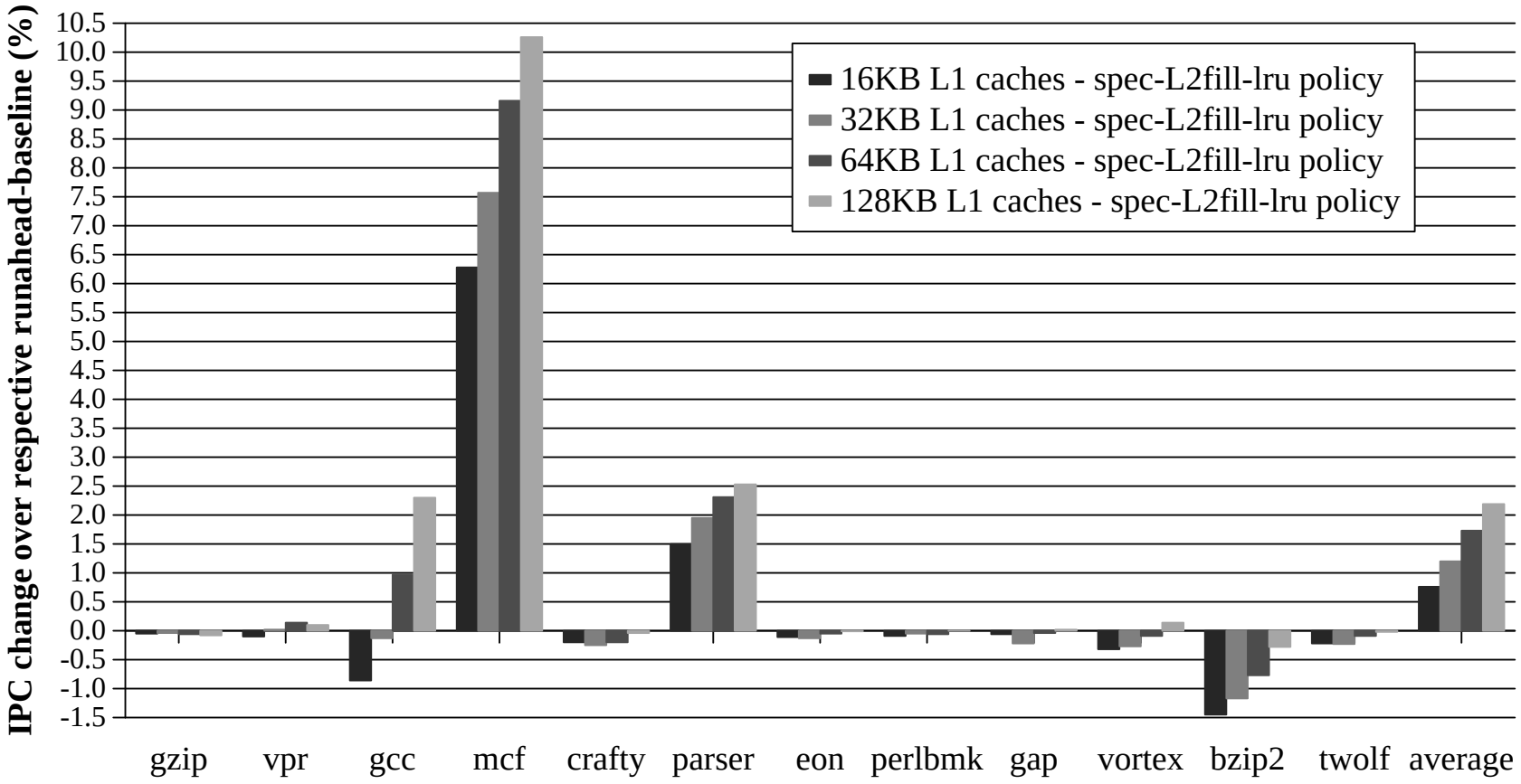
- spec-L2fill-lru is a better filtering policy in the presence of prefetcher references.
  - Many prefetched blocks are needed by correct execution shortly after they are evicted from L1 cache.

- Filtering is more effective for wrong-path references than for prefetcher references
  - If wrong-path references are not needed while they are in the L1 cache, they are more likely to be never needed than prefetcher references.
  - More analysis and data in the paper (Section 3.1)

Sensitivity to L1 cache size (*Stream-baseline)*



Sensitivity to L1 cache size (*Runahead-baseline)*



Conclusions

- Negative performance impact of speculative references is mainly due to L2 cache pollution.
- Using the L1 cache as a filter to reduce the L2 cache pollution is effective.
- Filtering policies are more effective for wrong-path references than for prefetcher references.
- The bigger the filter (L1 cache), the more effective the filtering policies.