Prefetching

Getting ahead of the cache miss curve

Prefetch

- Accesses often follow simple patterns
- If the pattern can be identified, lines can be fetched before they are actually needed, avoiding misses
- Danger of fetching too soon -- may be evicted
- Can evict useful data and force additional misses
- Can generate excessive memory traffic

Wang   ISCA 2003

Guided Region Prefetching: A Cooperative Hardware/Software Approach

L2 Cache Prefetch

- RAM access time more costly than L1 or L2
- L2 has more idle time, allowing more costly prefetch
- Longer lines carry more information to work from
- L2 to RAM prefetch in a multiprocessor can impede shared memory access

Scheduled Region Prefetch

- Aggressive prefetch of 4KB blocks on L2 miss
- Prefetch engine queues prefetch requests
- Prioritizer prevents interference with L2 misses
- More recent requests take priority
- Prefetch replaces LRU block
- Increases memory traffic by 180%

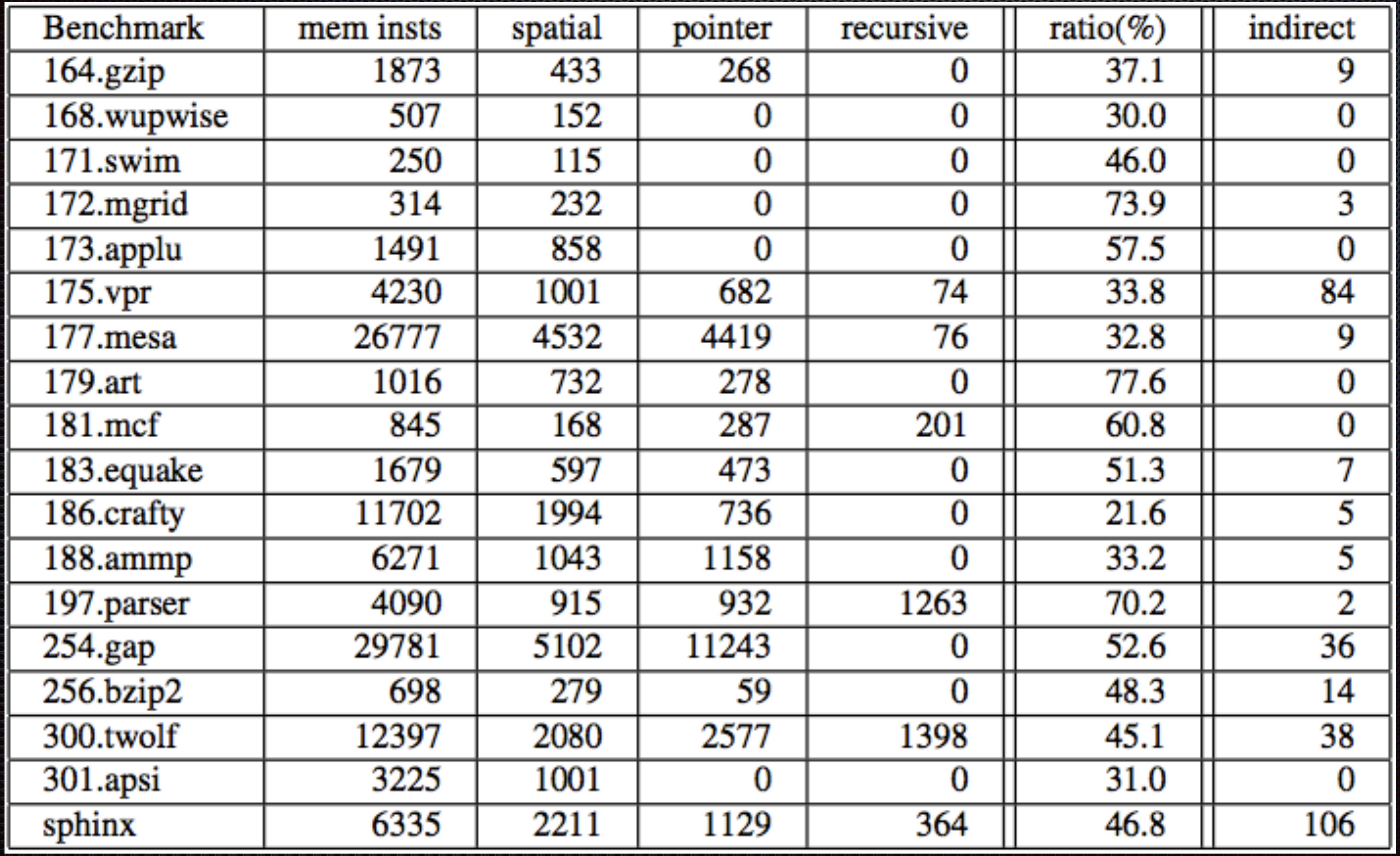
Guided Region Prefetch

- Add pointer prefetching: assume any value that falls within the address range of the heap is a pointer and prefetch that address if not already in cache
- Recursively prefetch “pointer” values found within a prefetched line to a depth of six
- Constrain size of prefetch using loop bounds info
- Use compiler hints to improve accuracy and reduce number of generated prefetches

Compiler Hints

- Spatial: tags a load as likely to need spatial prefetch (leaves many loads untagged)
- Size: indicates range to prefetch
- Indirect: assume miss location is part of an index array and prefetch using the line's values as indexes
- Pointer: the miss is getting data in a structure that contains pointers, so use those to prefetch
- Recursive pointer: the miss is part of a pointer chain

Results: Number of Hints



Benchmark	mem insts	spatial	pointer	recursive	ratio(%)	indirect
164.gzip	1873	433	268	0	37.1	9
168.wupwise	507	152	0	0	30.0	0
171.swim	250	115	0	0	46.0	0
172.mgrid	314	232	0	0	73.9	3
173.applu	1491	858	0	0	57.5	0
175.vpr	4230	1001	682	74	33.8	84
177.mesa	26777	4532	4419	76	32.8	9
179.art	1016	732	278	0	77.6	0
181.mcf	845	168	287	201	60.8	0
183.equake	1679	597	473	0	51.3	7
186.crafty	11702	1994	736	0	21.6	5
188.ammp	6271	1043	1158	0	33.2	5
197.parser	4090	915	932	1263	70.2	2
254.gap	29781	5102	11243	0	52.6	36
256.bzip2	698	279	59	0	48.3	14
300.twolf	12397	2080	2577	1398	45.1	38
301.apsi	3225	1001	0	0	31.0	0
sphinx	6335	2211	1129	364	46.8	106

Results: Pointer Prefetch

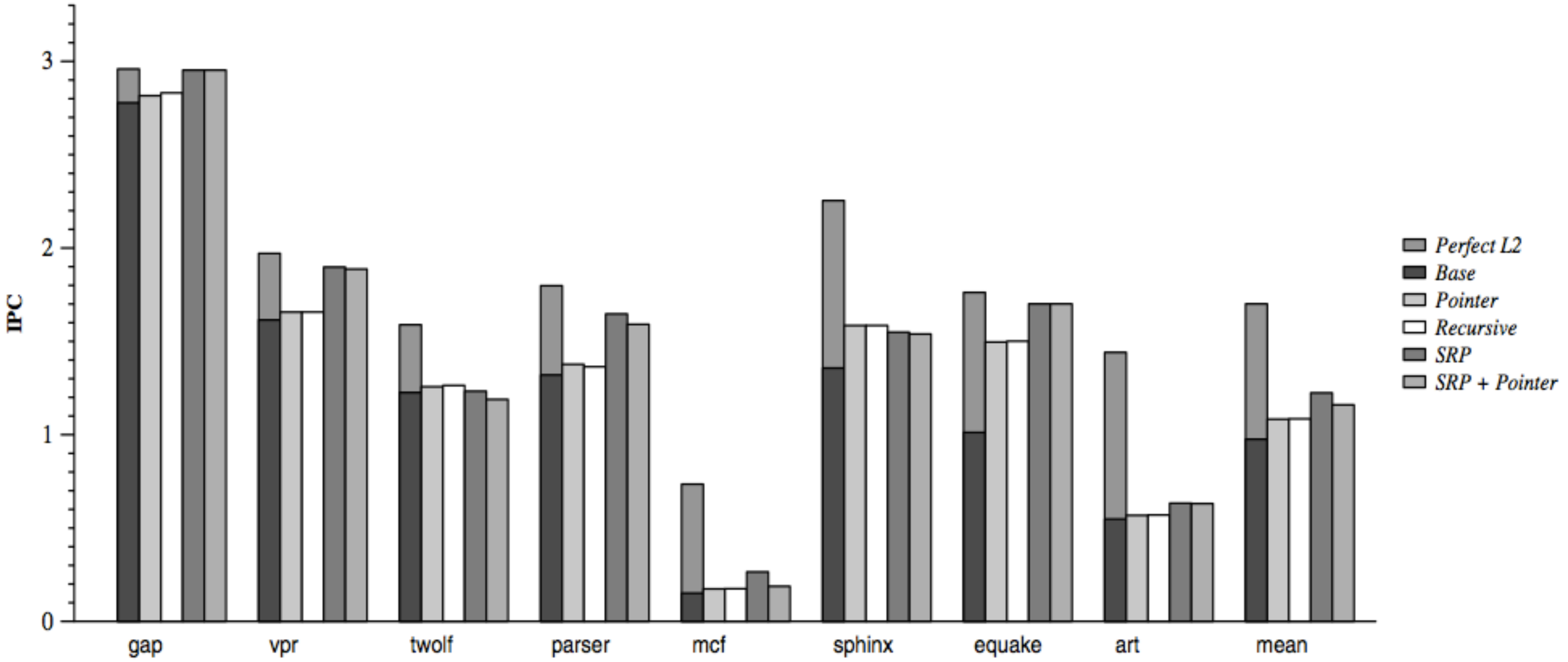


Figure 9: Performance gains from pointer prefetching

Comparison: Stride, SRP

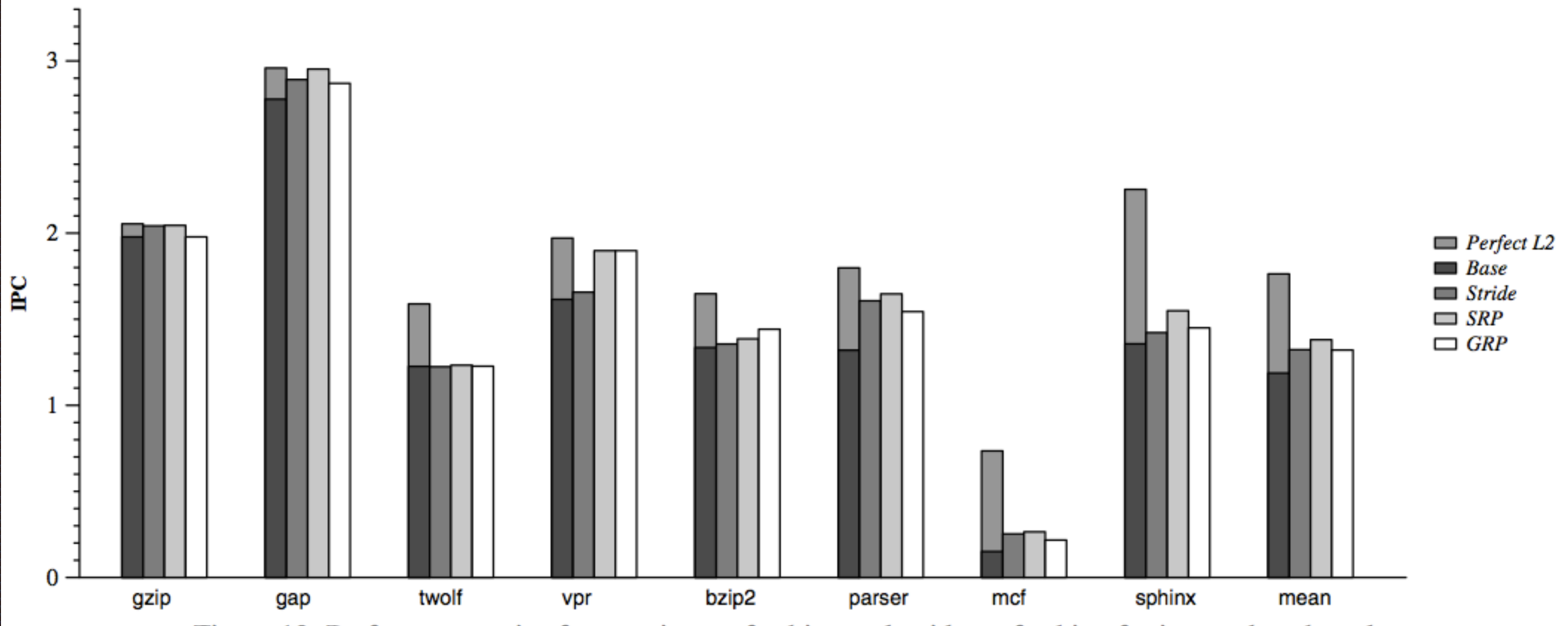


Figure 10: Performance gains from region prefetching and stride prefetching for integer benchmarks

Stride prefetcher has more expensive hardware

Same, but for FP

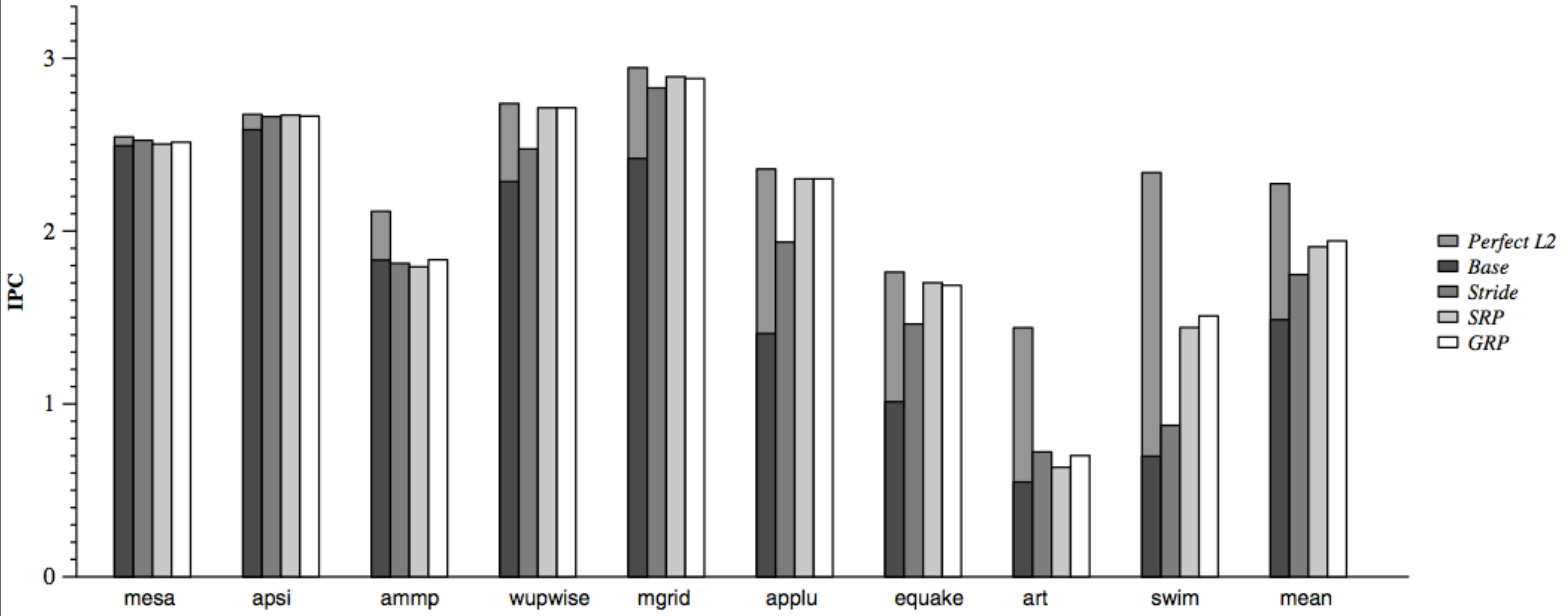


Figure 11: Performance gains from region prefetching and stride prefetching for floating-point benchmarks

Bandwidth Effect

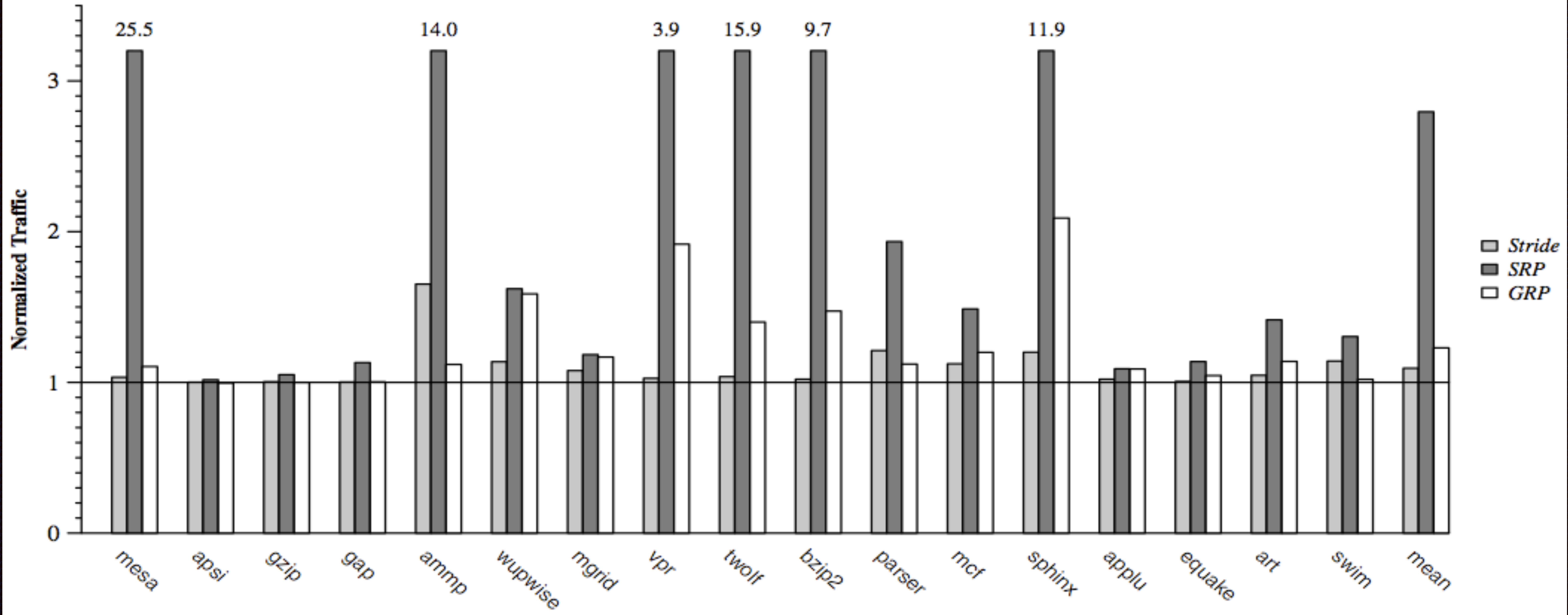


Figure 12: Normalized traffic

Discussion

Jain 18

Rethinking Belady's Algorithm to Accommodate Prefetching

Belady’s Algorithm (MIN)

- Originally for OS page replacement
- Peer into the future
- Replace page that will not be used for the longest time
- Because page mapping is fully associative, there is no specification for what to do when there are multiple lines that are no longer needed
- Obviously not practical

Terminology

- Demand — the processor requests a location
- Demand miss — the request misses, Demand hit — the request hits
- Prefetch — a separate predictor tries to anticipate a demand
- Accurate prefetch — one that is demanded before eviction
- Shadowed — a prefetch followed by another without a demand between
- Prefetch friendly — a successful prefetch following a demand
- Dead — a demand load followed by an inaccurate prefetch

Basic Idea

- Optimizing demand misses ignores effect of prefetch
- Change policy to first evict the line that will be prefetched furthest in the future, then fall back to evicting the line with a demand request furthest in the future
- Prefetches can be inaccurate, and they can come too early
- The idea is to evict things that will be prefetched later to make more room for demand fetches

Example

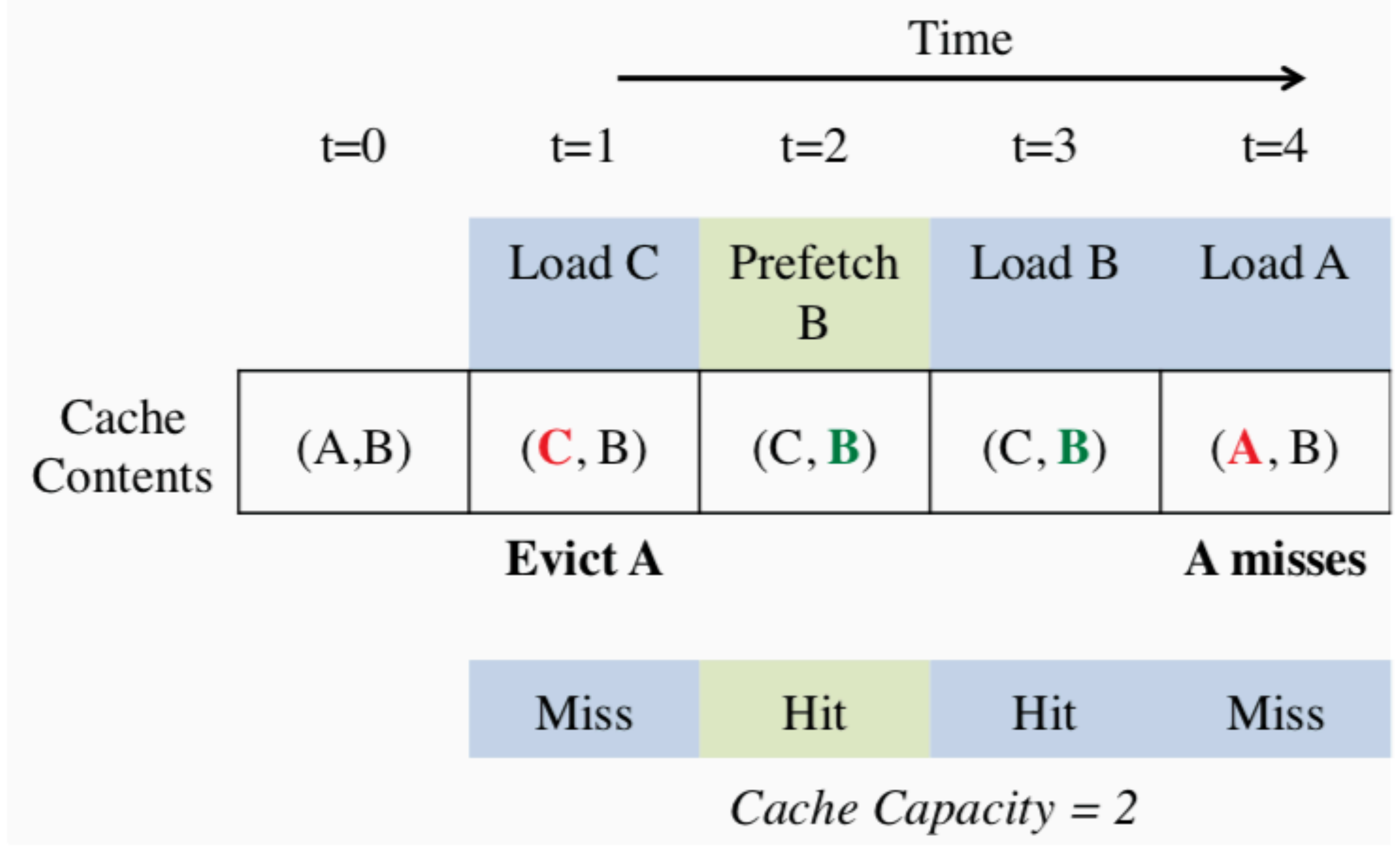


Figure 3. Belady's MIN results in 2 demand misses.

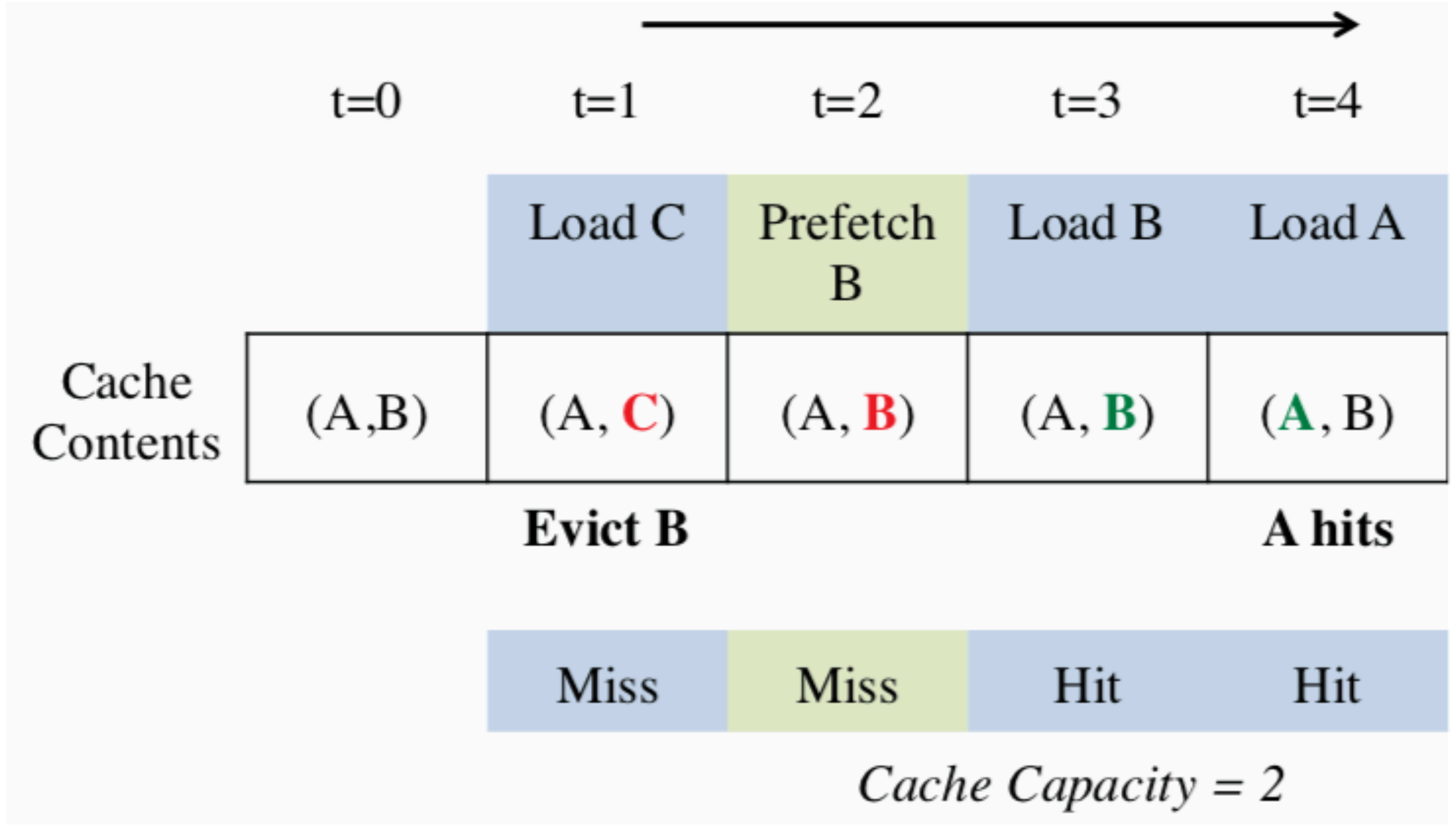
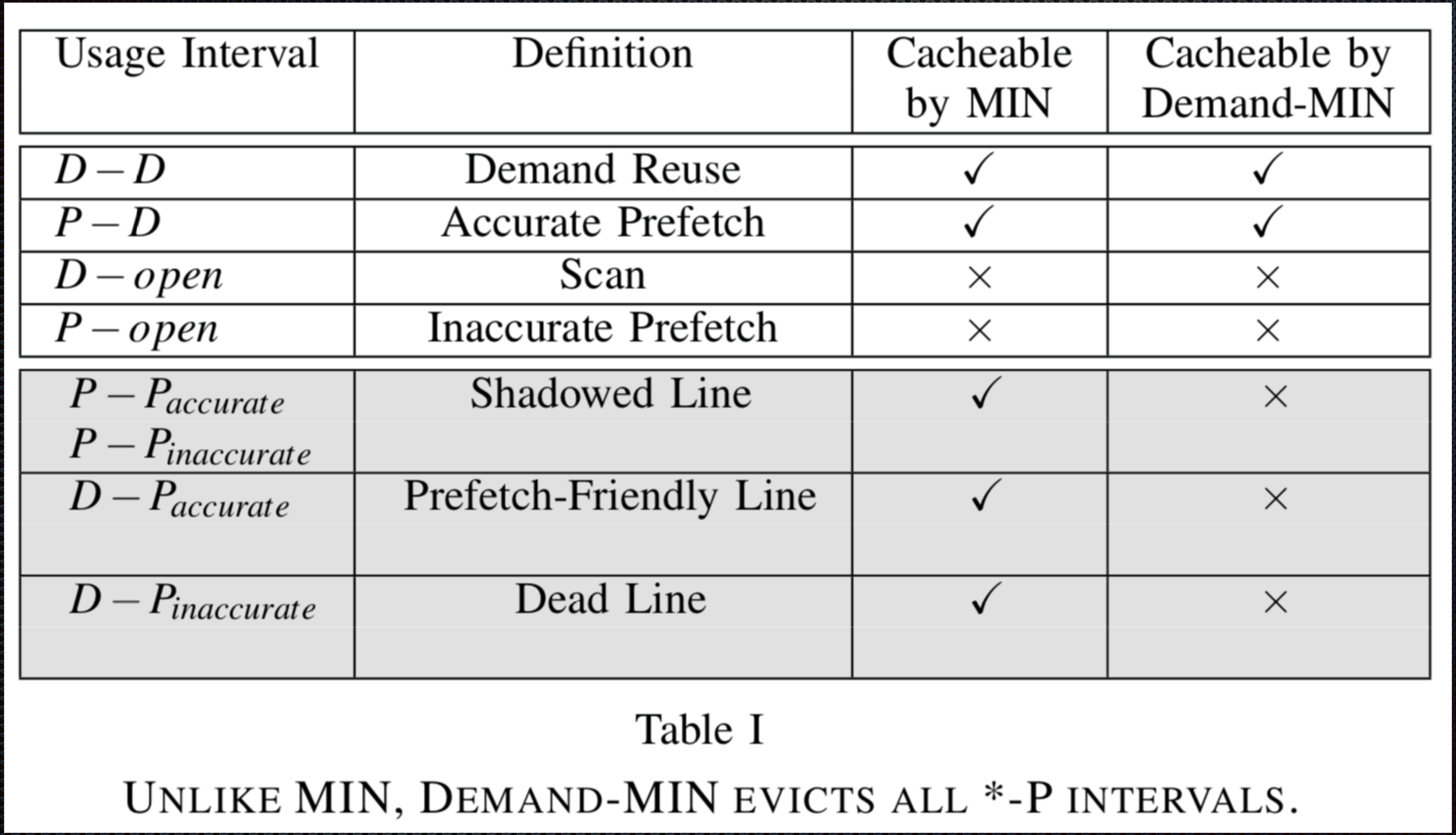


Figure 4. Demand-MIN results in 1 demand miss.

Evict A because it is demanded furthest ahead

Evict B because it is going to be prefetched again anyway

Line Usage Intervals



Usage Interval	Definition	Cacheable by MIN	Cacheable by Demand-MIN
$D-D$	Demand Reuse	✓	✓
$P-D$	Accurate Prefetch	✓	✓
$D-open$	Scan	×	×
$P-open$	Inaccurate Prefetch	×	×
$P-P_{accurate}$ $P-P_{inaccurate}$	Shadowed Line	✓	×
$D-P_{accurate}$	Prefetch-Friendly Line	✓	×
$D-P_{inaccurate}$	Dead Line	✓	×

Table I

UNLIKE MIN, DEMAND-MIN EVICTS ALL *-P INTERVALS.

Avoids wasting space on prefetches that aren’t needed or will recur

Demand-MIN

- Still requires an oracle
- Produces a schedule that minimizes demand misses with prefetch
- However, it can produce more memory traffic because it evicts more prefetched values which then need to be prefetched again

Examples

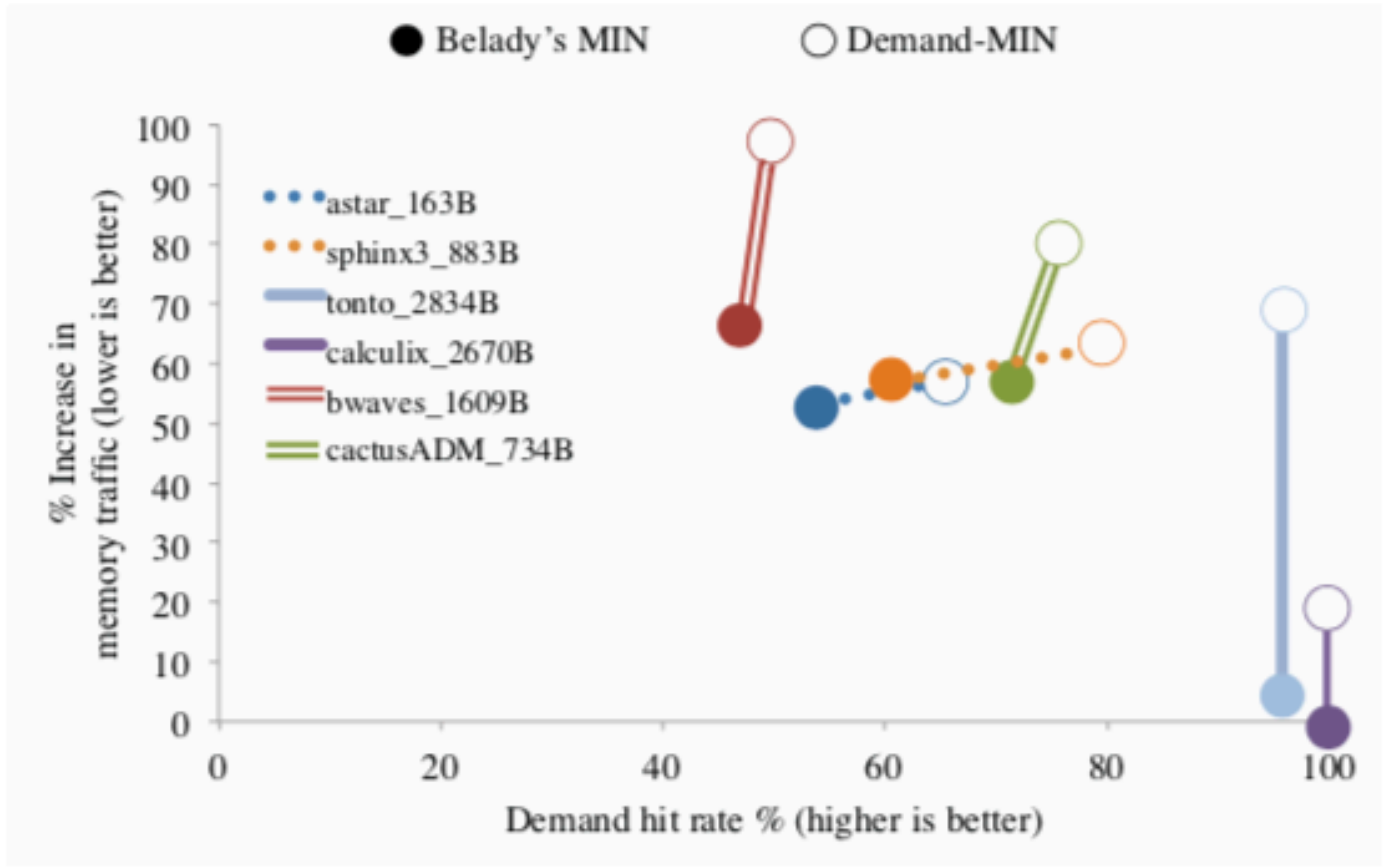


Figure 2. With prefetching, replacement policies face a tradeoff between demand hit rate and prefetcher traffic.

Need Something Adjustable

- Flex-MIN — Evict the line that is prefetched furthest in the future and is not protected, otherwise fall back to MIN
- Protected lines — At the beginning of *-P intervals, and if evicted would increase traffic with little improvement in demand hit rate
- Example: Evicting a long chain of short-duration prefetched values to keep a demand value in cache, vs. evicting a smaller number of long-duration prefetched values

Hawkeye cache

- Learns behaviors of references based on PC value
- Approximates Optimal by looking for next reuse, later reuses are better eviction candidates, which would have been what OPT would predict
- Associate the learned prediction with the PC value
- Claim it can be implemented efficiently (it hasn’t been implemented)
- 2K 5-bit counters per core

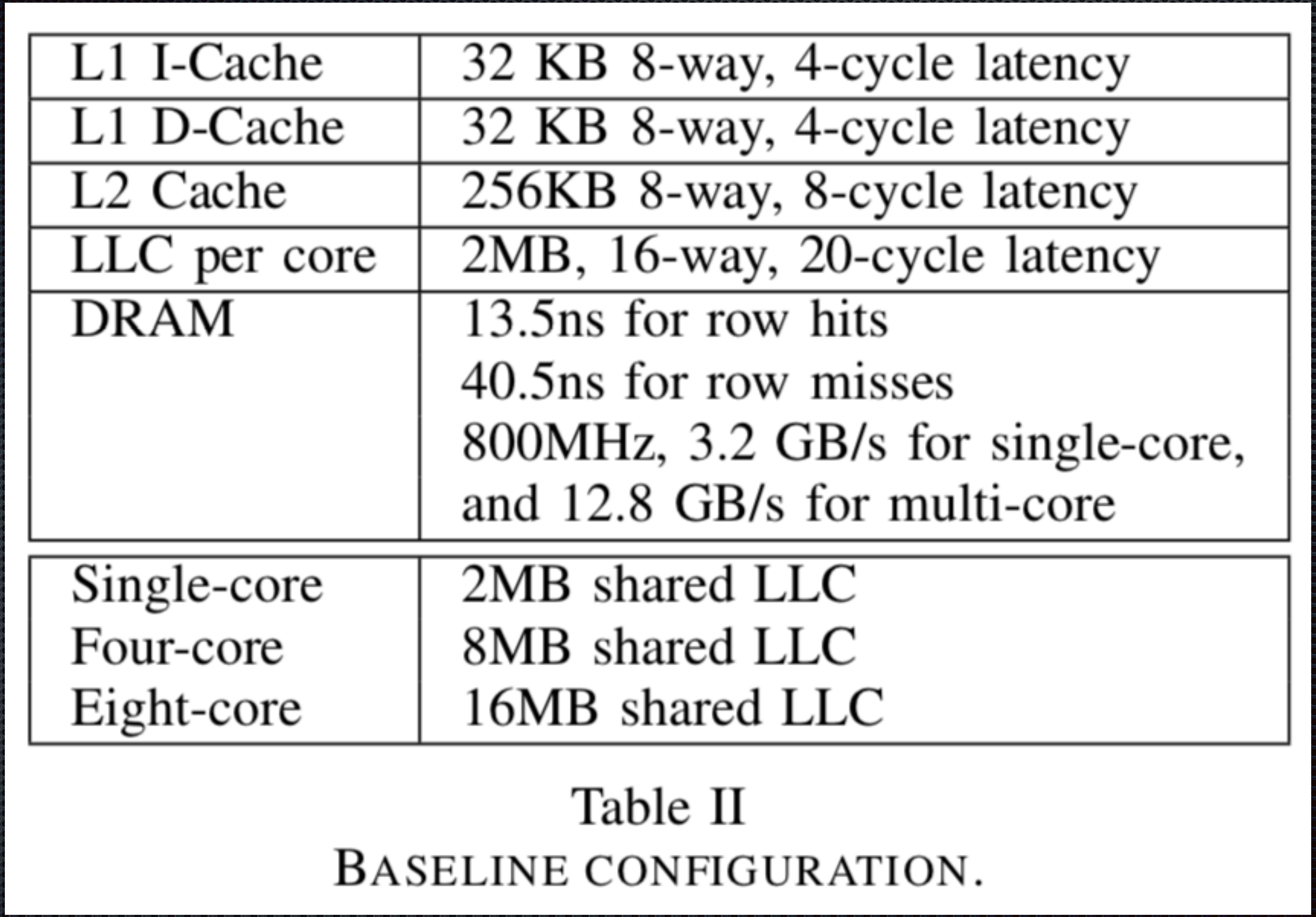
Harmony

- Changes Hawkey to learn from Flex-MIN instead of MIN
- Distinquishes between *-P and *-D intervals
- Only allows *-P caching if length is less than a threshold (keep the short duration prefetches to avoid extra memory traffic)
- Threshold is ratio of D-D intervals tha this to *-P intervals that are cache friendly (Lines Evicted Per Demand Hit or LED)
- Keeps four counters for Demand Miss and cache-friendly *-P statistics

Methodology

- Champ-SIM
- Trace-based, 6-wide OoO, RoB, 3-levels cache, reasonably sophisticated memory simulation, perceptron branch predictor, not validated
- L1 uses next-line prefetch (49% accurate), L2 uses stride prefetch (63%)
- Use SimPoint to select traces, at least 1B instructions with 200M warmup

Configuration



L1 I-Cache	32 KB 8-way, 4-cycle latency
L1 D-Cache	32 KB 8-way, 4-cycle latency
L2 Cache	256KB 8-way, 8-cycle latency
LLC per core	2MB, 16-way, 20-cycle latency
DRAM	13.5ns for row hits 40.5ns for row misses 800MHz, 3.2 GB/s for single-core, and 12.8 GB/s for multi-core
Single-core Four-core Eight-core	2MB shared LLC 8MB shared LLC 16MB shared LLC

Table II
BASELINE CONFIGURATION.

1B instructions won't give statistical significance on 2MB LLC accesses

Results

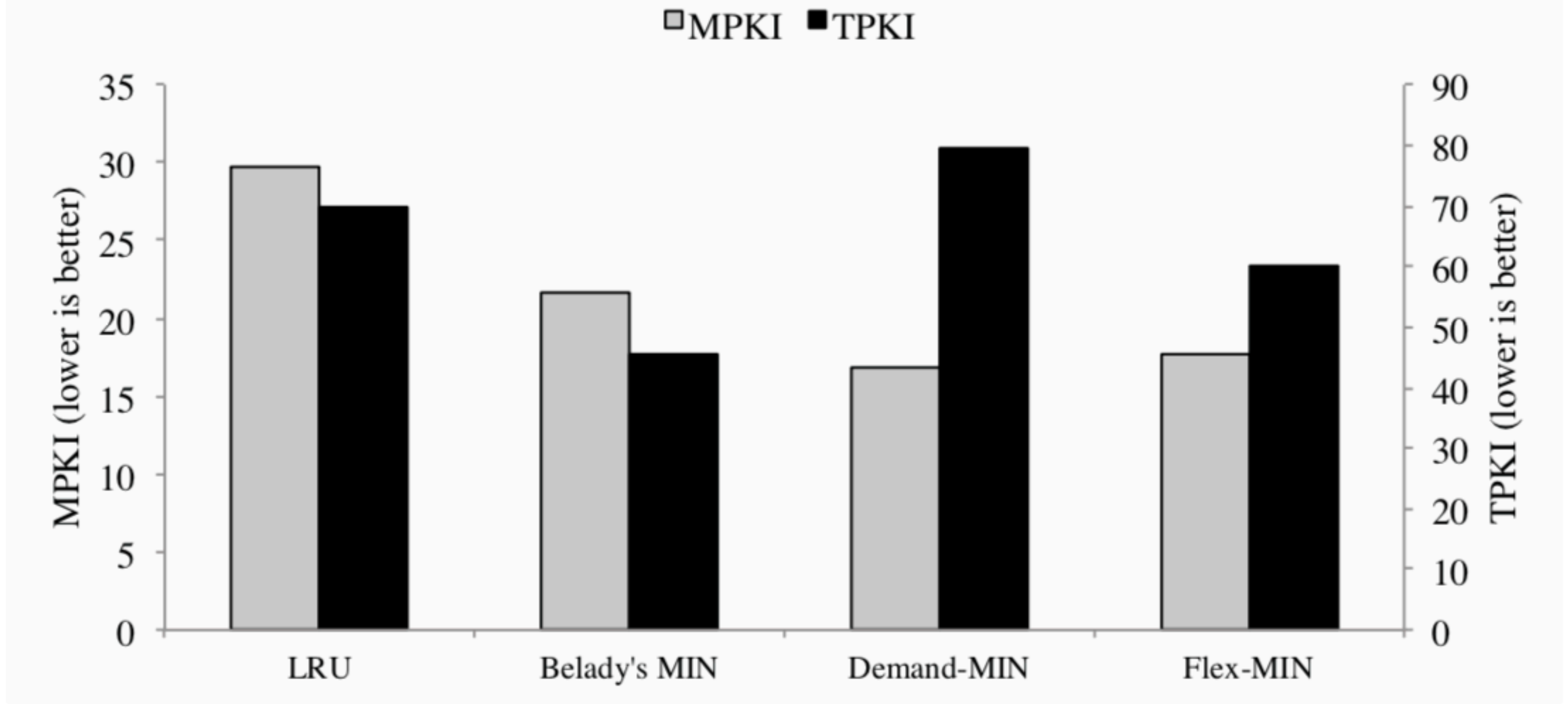


Figure 11. Flex-MIN achieves a good tradeoff between MIN and Demand-MIN.

Results for more cores

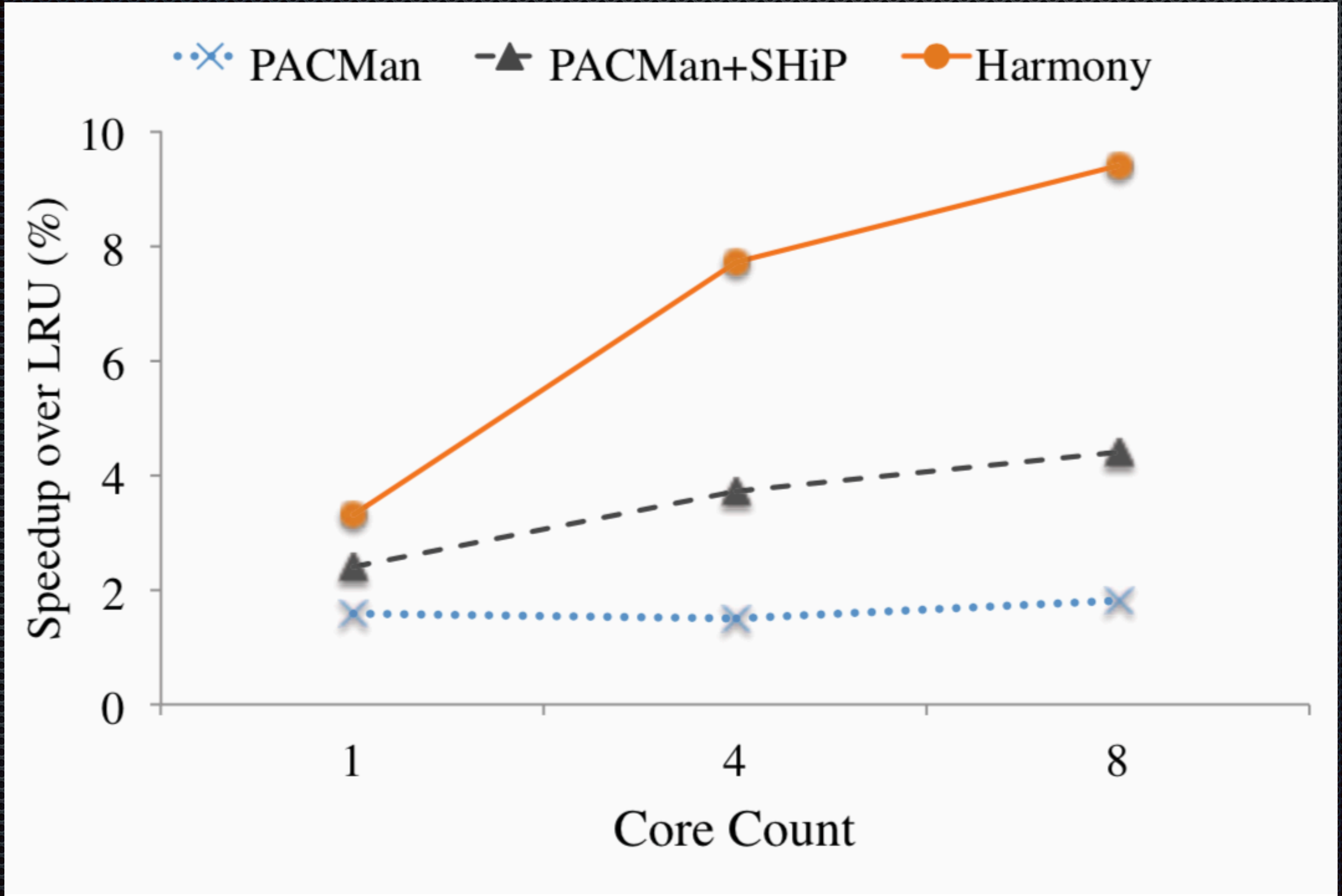


Figure 14. Harmony's advantage increases with more cores.

Harmony Variants WRT LRU

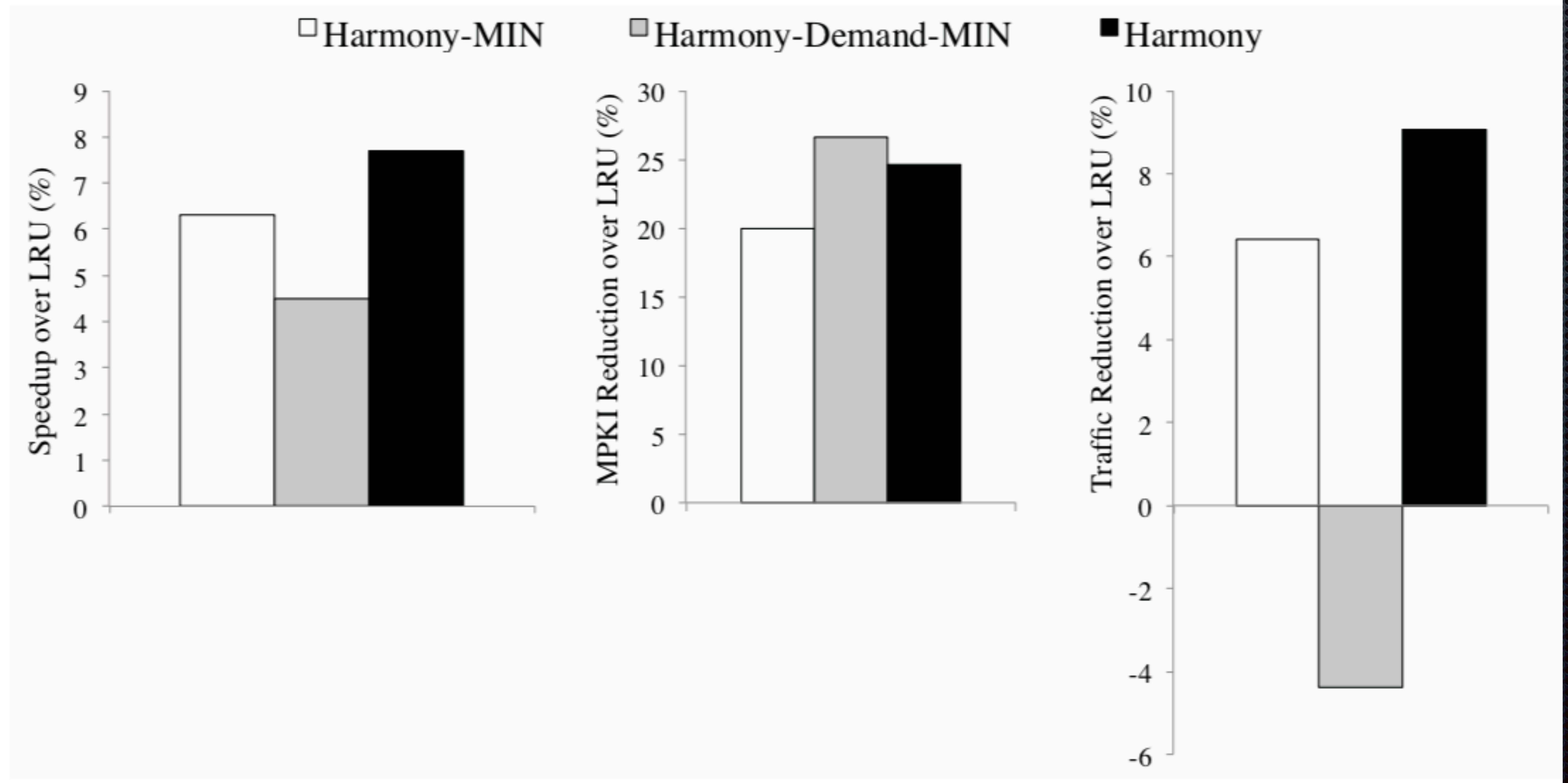


Figure 17. Three Versions of Harmony.

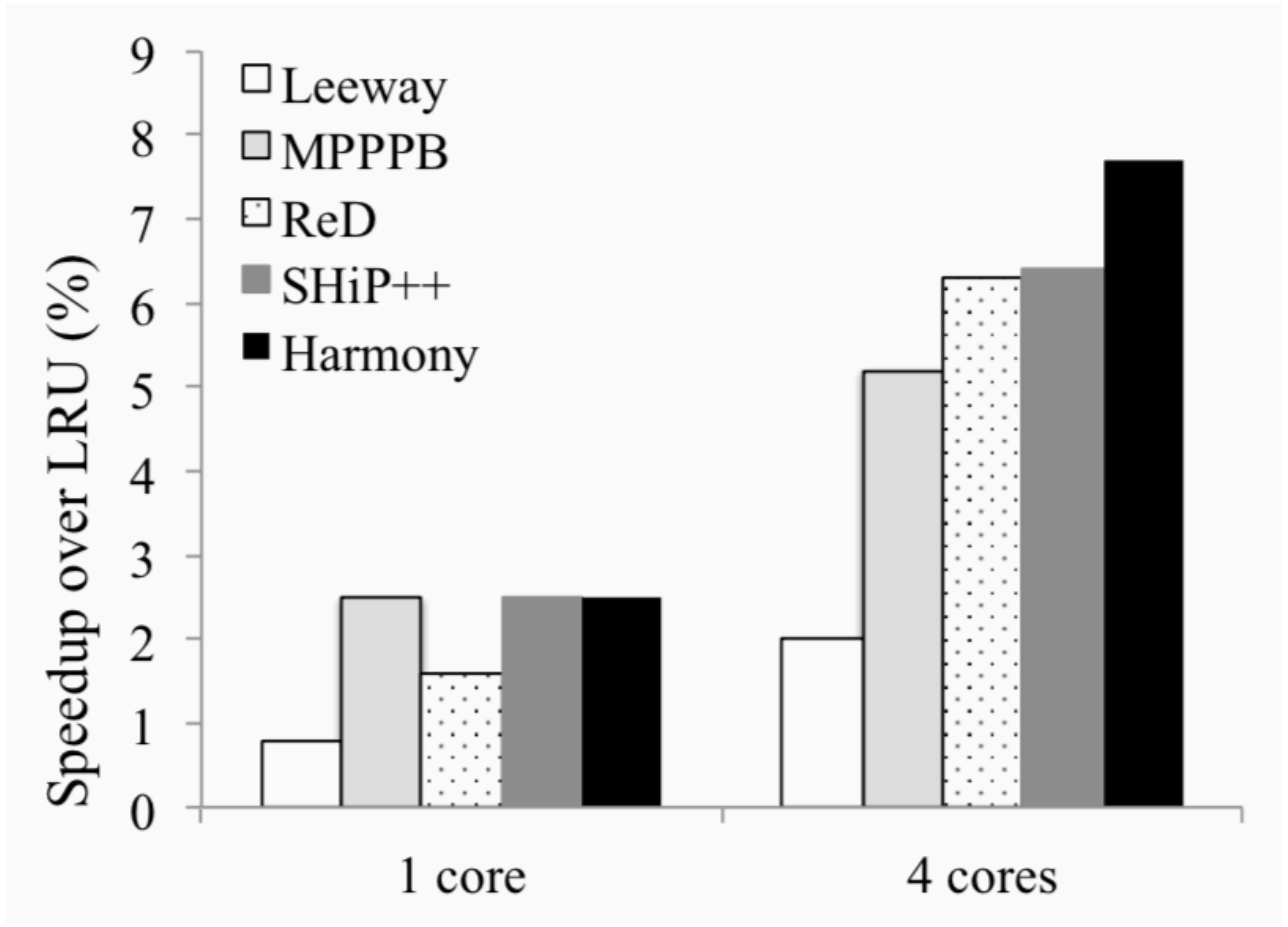


Figure 19. Harmony outperforms top submissions to the Cache Replacement Championship 2017. An older version of Harmony won the championship.

But it won a contest!

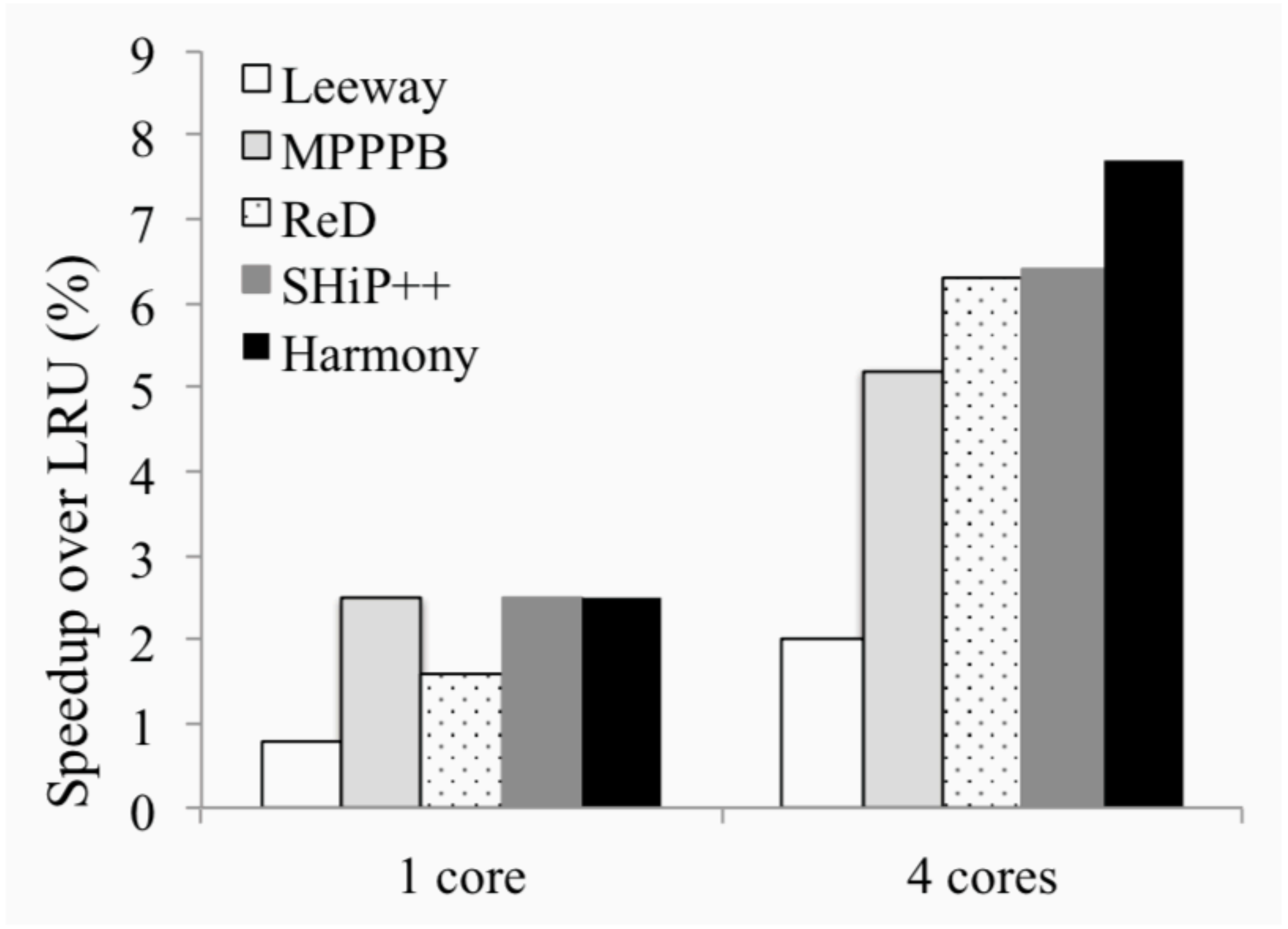


Figure 19. Harmony outperforms top submissions to the Cache Replacement Championship 2017. An older version of Harmony won the championship.

Discussion