Prefetching-Aware Cache Line Turnoff for Saving Leakage Energy

Ismail Kadayif
Dept of Computer Engineering,
Canakkale Onsekiz Mart University, Turkey
kadayif@comu.edu.tr

Mahmut Kandemir and Feihui Li
Dept of Computer Science and Engineering,
Pennsylvania State University, U.S.A
{ kandemir,feli }@cse.psu.edu

Road Map

- Background
- Objective
- Prefetching and Cache Line Turnoff
- Experimental Setup
- Base Results
- Different Cache Line Turnoff Schemes
- Their Experimental Results
- Summary and Future Work

Cache Memories

- They are critical components in system design
  - performance
    - hit/miss characteristics affect the performance
  - energy consumption
    - they are responsible for a significant portion of system energy consumption

Background

- Performance and energy optimization strategies have been developed separately
- Performance oriented techniques
  - May introduce energy overhead
  - Latency-reducing compiler optimisations [A. Gupta et al. ISCA'91]
  - Customized cache hierarchy design [S. Cotterel and F. Vahid ICCAD'02]
  - Latency-hiding techniques such as prefetching [R. Cooksey et al. ASPLOS'02, A. C. Lai et al. ISCA'01]

Background Cont.

- Leakage Energy optimization techniques
  - May cause performance degradation
  - State destroying
    - Cache decay [S. Kaxiras et al. ISCA'01]
  - State preserving (data retaining)
    - Drowsy caches [K. Flautner et al. ISCA'02]
  - Hybrid
    - State destroying + state preserving [L. Li et al. PACT'02]

Objective

- Interactions between performance-oriented and energy-oriented optimizations have not been studied much.
- Study these interactions and show how performance and energy optimizations affect each other.
- We use prefetching and cache line turnoff for performance and leakage energy optimization, respectively.

Prefetching

- Bring data/instructions from main memory to cache before they are actually needed.
- In this study we used hardware-based technique called *tagged prefetch* which was implemented on HP PA7200.
  - The prefetch is initiated for block b + 1 when block b is accessed under two different scenarios:
    - If there is a miss when b is brought into memory.
    - If b is brought into cache via prefetching and it is accessed for the first time.

Cache Line Turnoff

- For L1 cache we implemented *cache decay* originated from S. Kaxiras et al [ISCA'01].
  - A cache block which is not used for a sufficiently long period of time is put into state destroying (SD) low power mode to save leakage energy.

Cache Line Turnoff

- For L2 cache we implemented *S-SP-Lazy (Speculative, State-Preserving, and Lazy)* mechanism originated from L. Li et al [PACT'02].
  - When data is brought from L2 to L1, the corresponding L2 subblock is put in state preserving (SP) leakage control mode.

Experimental Setup

- Prefetching and leakage optimizations were applied in both I-cache and D-cache
- We used SimpleScalar 3.0 tool to implement prefetching and leakage-saving optimization strategies
- Cacti 3.2 tool set for dynamic energy per access
- Different leakage factors represented by k (the ratio between the leakage energy per cycle of entire cache and the dynamic energy consumed per access)
- Randomly-selected benchmarks from SPEC2000 suit
  - Fast forward 300 million instructions and then simulate the next 200 million instructions

Power States

- A cache line can be in one of the three power modes
  - Active (AC) mode: consuming full power.
  - State Preserving (SP) mode: the cache line consumes 10% of the leakage power of active mode.
  - State Destroying (SD) mode: no power consumption at all.

Base Configuration

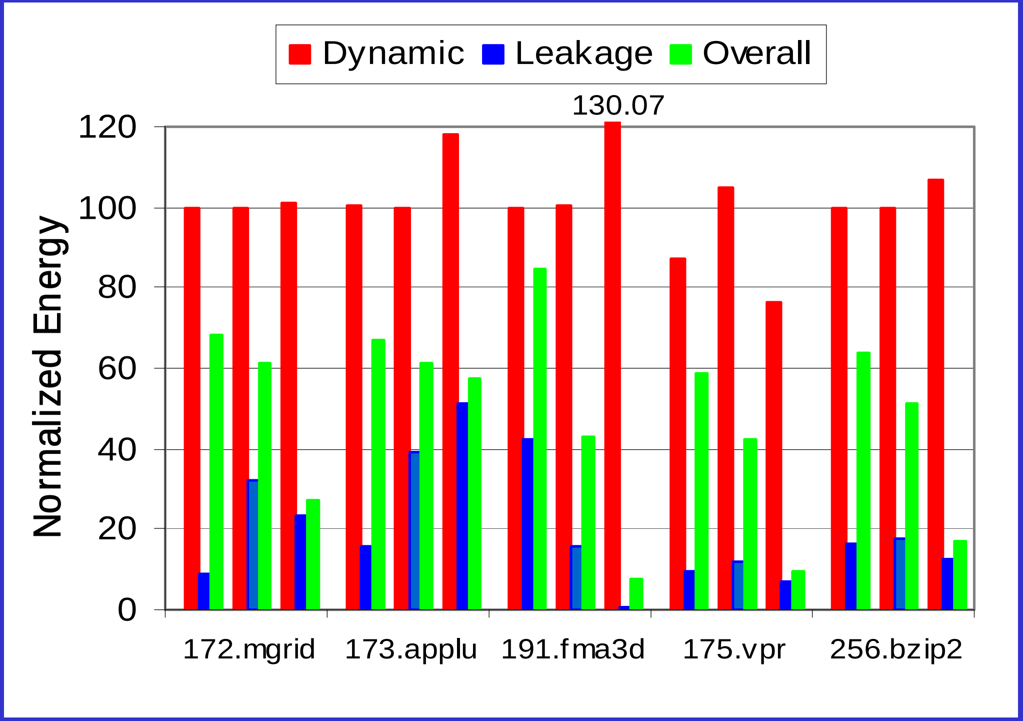
Processor Core	
Functional Units	4 integer and 4 FP ALUs 1 integer multiplier/divider 1 FP multiplier/divider
LSQ Size	64 Instructions
LRU Size	64 Instructions
Fetch/Decode/Issue/Commit Width	4 instructions/cycle
Fetch Queue Size	4 instructions
Cache and Memory Hierarchy	
L1 Instruction/Data Cache	32KB, 32 byte blocks, 2-way, 1cycle latency
L2 Cache	1MB unified, 128 byte blocks, 2-way, 10 cycle latency
Data/Instruction TLB	128 entries, full-associative, 30 cycle miss latency
Memory	100 cycle latency
Energy Management	
Technology	0.07 micron
Supply Voltage	1.0V
Dynamic Energy per L1 Access	0.186nj
Dynamic Energy per L2 Access	0.871nj
Leakage Energy per L1 Block/ L2 Subblock per Active Cycle	0.182pj
Leakage Energy per L2 Subblock per Standby Cycle (state preserving)	0.018pj
Leak.En. per L1 block/L2 Subblock per Standby Cycle (state-destroying)	0pj

Benchmarks and Their Salient Characteristics

Benchmarks	Execution Cycles (millions)	L1 Instruction Cache Energy (mJ)		L1 Data Cache Energy (mJ)		Unified L2 Cache Energy (mJ)		Total Leakage Energy (%)
		Dyn.	Leak.	Dyn.	Leak.	Dyn.	Leak.	
172.mgrid	106.25	37.34	19.76	14.34	19.76	5.31	632.40	92.18%
173.applu	131.98	37.43	24.54	15.34	24.54	12.16	758.28	92.78%
191.fma3d	82.81	43.25	15.40	7.13	15.40	1.01	492.80	91.06%
175.vpr	159.63	52.23	29.69	15.13	29.69	5.28	950.08	93.29%
256.bzip2	115.70	37.25	21.52	15.74	21.52	5.63	688.64	92.58%

Leakage energy of L1-L2 cache hierarchy dominates its dynamic energy due to relatively large L2-cache size and few L2-cache accesses.

Normalized Energy Consumption of Optimized Codes (k=1) (Base Results)



- We used the the decay periods used by L. Lee et al. [PACT'02] (10K cycles for L1 and 1M cycles for L2) to put cache blocks/sub-blocks in SD mode.
- The first and second group of bars are for the L1 instruction, L1 data caches, respectively, while the last group is for the L2 cache.
- 191.fma3d has the largest dynamic energy increase by around 30% for the L2 cache.

Normalized Execution Cycles of Optimized Codes (Base Results)

172.mgrid	173.applu	191.fma3d	175.vpr	256.bzip2
65.07%	76.08%	118.22%	54.67%	67.83%

We have, on average, 21.62% performance improvement.

Customizing Cache Line Turnoff

- In base experiments, all the cache lines are treated exactly the same way (regardless of being brought into the cache through prefetching or through a normal load operation).
  - A prefetched cache line is placed in SD power mode if it is not touched during the decay period.

Customizing Cache Line Turnoff

- Three different optimization strategies
  - Treat the prefetches cache lines differently from the normal (non-prefetched) cache lines.
- S-SP (Speculative and State Preserving)
  - The prefetched cache lines are put in SP mode immediately after the prefetching.
- L-SD (Lazy and State Destroying)
  - It uses a much shorter decay period for the prefetched cache lines.
- P-H (Predictive and Hybrid)
  - Treats a prefetched cache line based on the characteristics of the previous prefetch into the same line.

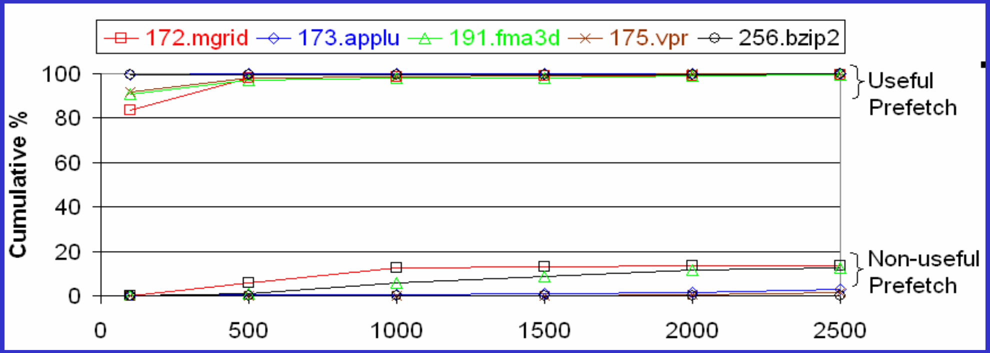
S-SP Scheme

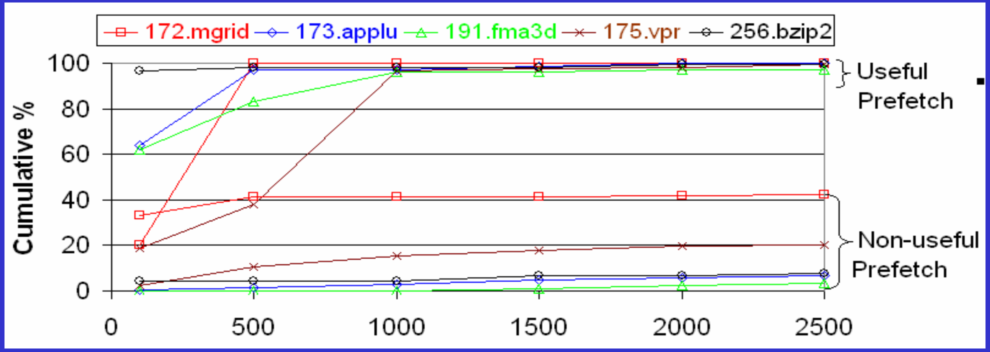
- Prefeched cache lines are placed in SP leakage control mode immediately after the prefetching is complete.
  - Advantage
    - Unnecessarily prefetched cache line (the cache line would be discarded from the cache without being accessed at all.) will be in the SP mode until it is being replaced, instead of being in the AC mode consuming full leakage.
  - Disadvantage
    - When the useful prefetched cache line is accessed, the line needs one extra cycle to be waken up (performance degradation).

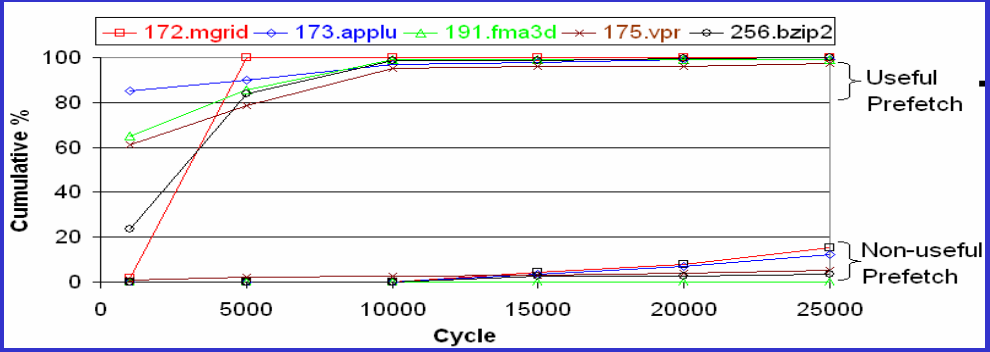
L-SD Scheme

- A new decay period for the prefetched cache lines.
  - Basic assumption: the useful prefetched cache lines, in general, is accessed within a short period of time after they are brought into the cache.
  - Put the prefetched cache line into the SD mode if it is not accessed within this short decay period.

The Cumulative Distribution of the Access Intervals for the Prefetched lines







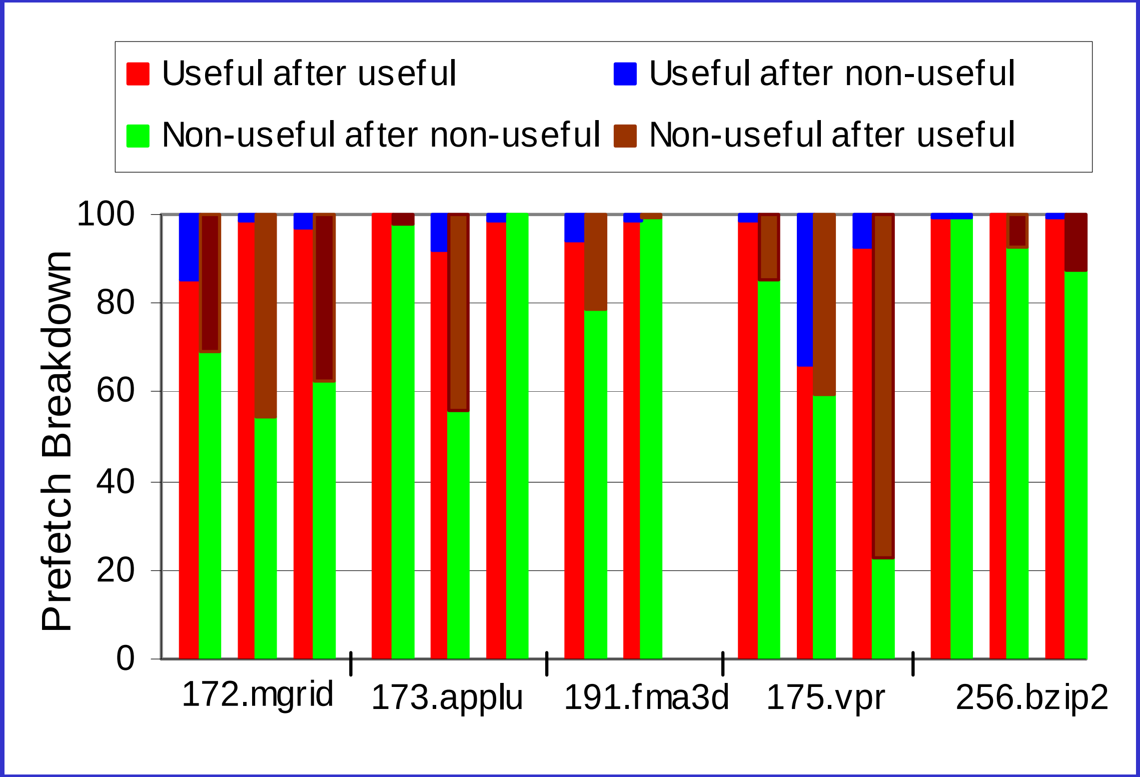
-Top: L1 instruction cache
-Middle: L1 data caches
-Bottom: L2 cache.

- Observations
  - -useful prefetched lines accessed within very short interval after being prefetched.
  - -non-useful prefetched lines spend considerable time before being discarded.
- Based on these observations, we selected the decay period of the prefetched cache lines as 1K cycles for the L1 instruction and data caches, and 10K cycles for the L2 cache.

P-H Scheme

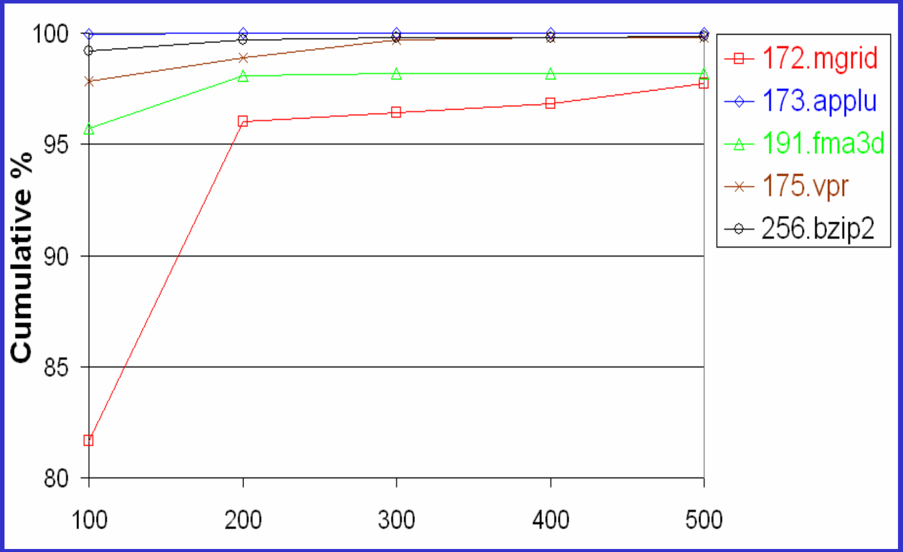
- Treats a prefetched cache line based on the characteristics of the previous prefetch into the same line.
- Predictive
  - It tries to figure out whether the prefetching into the line will be useful or not.
- Hybrid
  - The power status of the prefetched cache line can be either AC or SP.

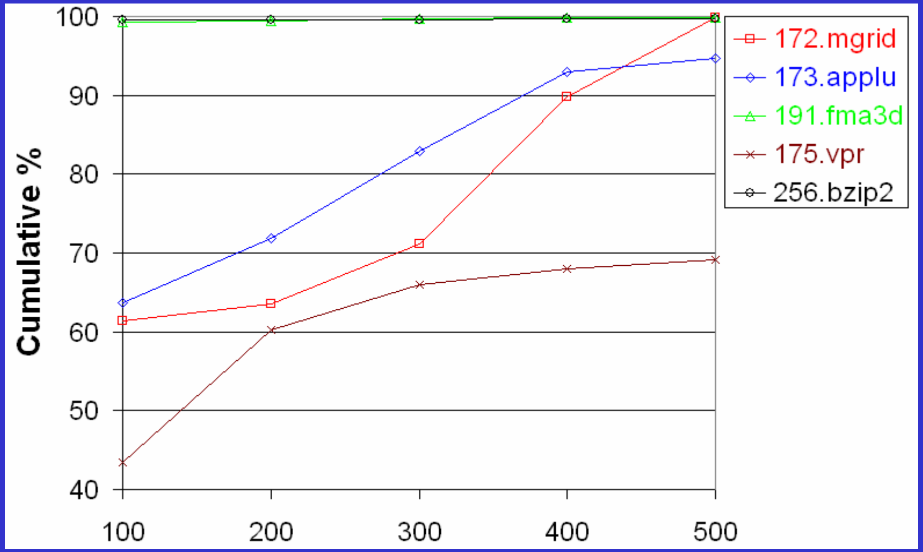
Breakdown of the Prefetches

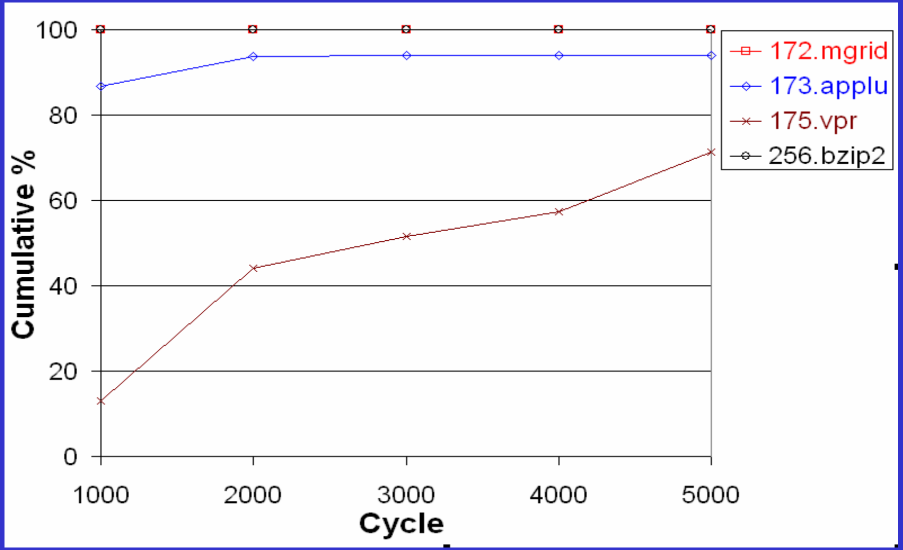


- Categorize the useful/non-useful prefetches based upon the previous prefetch into the same line
- Observations
  - If a prefetch is useful, the next prefetch into the same line will more likely be useful.
  - If a prefetch is non-useful, the next prefetch into the same line will more likely be non-useful.

Cumulative Distribution of the Differences Between the Access Intervals of the Two Successive Useful Prefetches into The Same Line

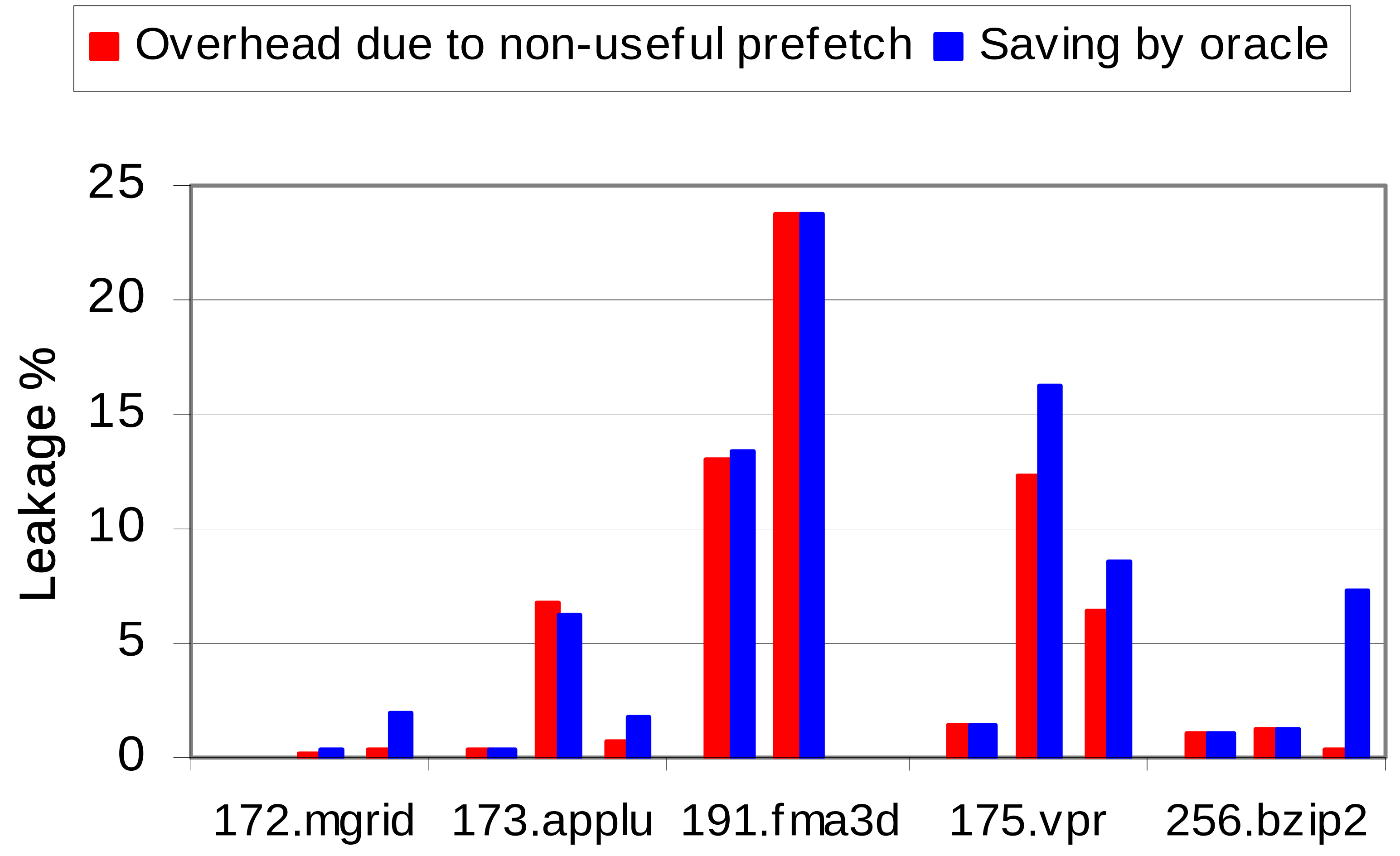




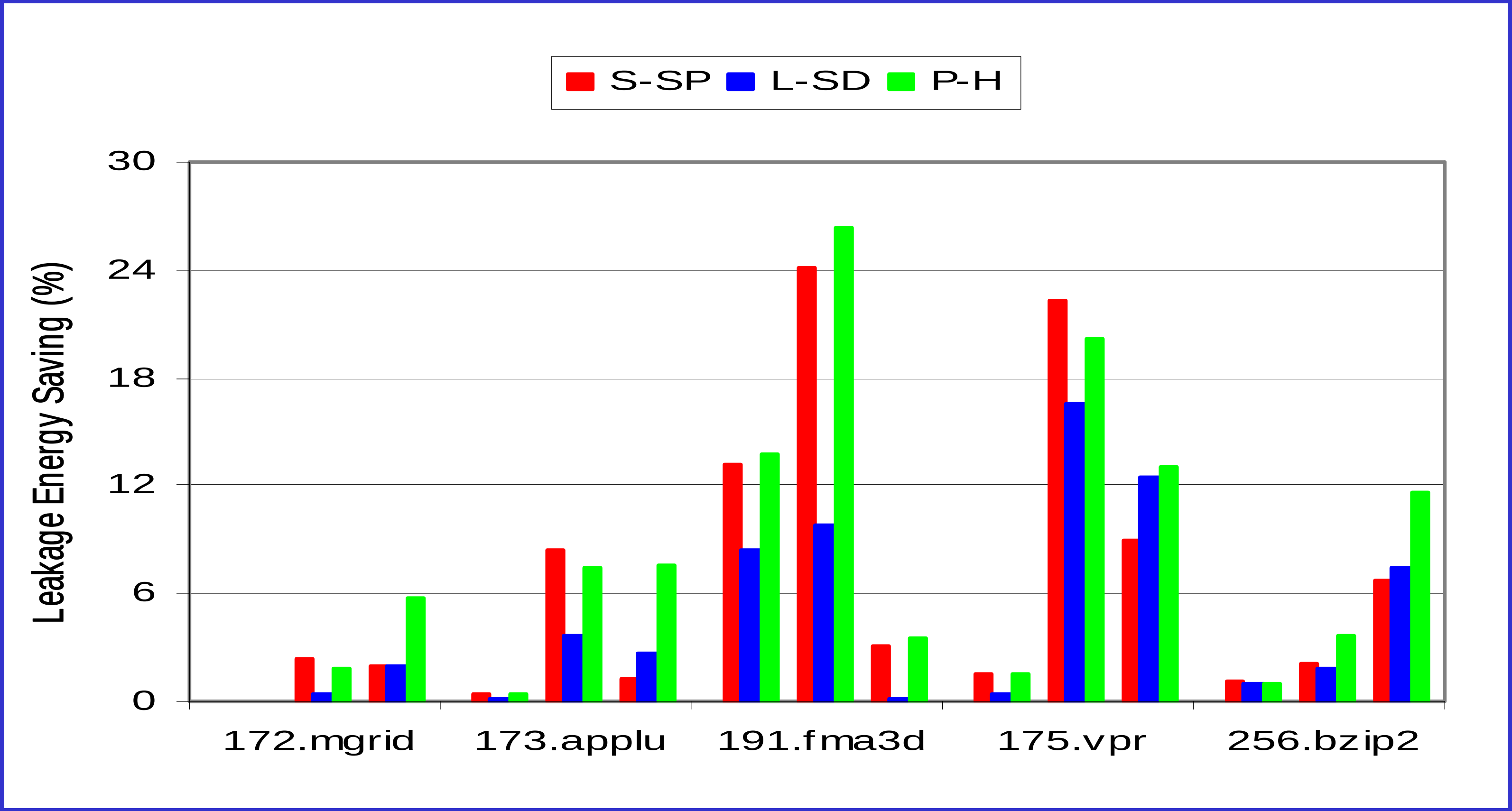


- Observation
  - The access interval of a prefetched cache line can be predicted quite accurately within a range.
- Threshold values for deciding the power status of the prefetched line
  - 200 cycles for both L1 instruction and data caches, 2000 cycles for L2 cache.

Leakage Energy Overheads Due to Non-useful Prefetches and the Leakage Savings by the Oracle



Leakage Energy Savings With Our Schemes



All the savings are as percentages w.r.t. leakage energy consumption of the optimized codes (base results), the codes
optimized by prefetching and leakage control mechanisms.
- S-SP outperforms L-SD in both L1 instruction and data caches.
- L-SD does a better job than S-SP in saving the L2 leakage energy.
- P-H outperforms L-SD for all caches.
- The average performance overhead is around 1% for each scheme, and their performance ranking is S-SP, P-H, and L-SD.

Summary & Future Work

- A study of cache energy and performance behavior when leakage optimization and prefetching are used together
- Three different cache line turnoff schemes
  - Prefetched lines are treated differently from those which are brought into the cache via normal load operations.
  - Increase leakage savings without compromising on performance
- Ongoing work
  - Investigate the effect of prefetching on soft error rate.