

Gated Precharging: Reducing Bitline Precharge in Deep-Sub μ Caches



Se-Hyun Yang and Babak Falsafi



PowerTap

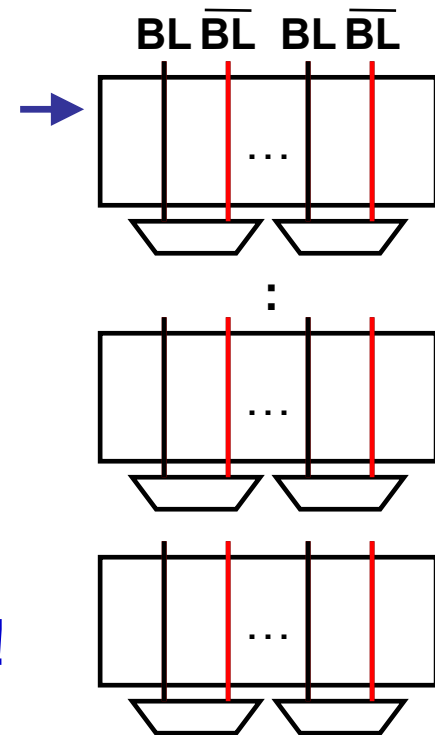
<http://www.ece.cmu.edu/~powertap>
Computer Architecture Lab (CALCM)
Carnegie Mellon University

High Bitline Leakage in Caches

Deep-sub μ high-performance caches

- ❑ Use subarrays
- ❑ Precharge entire cache
- ❑ Active subarrays: bitlines discharge
- ❑ Idle subarrays: bitlines leak

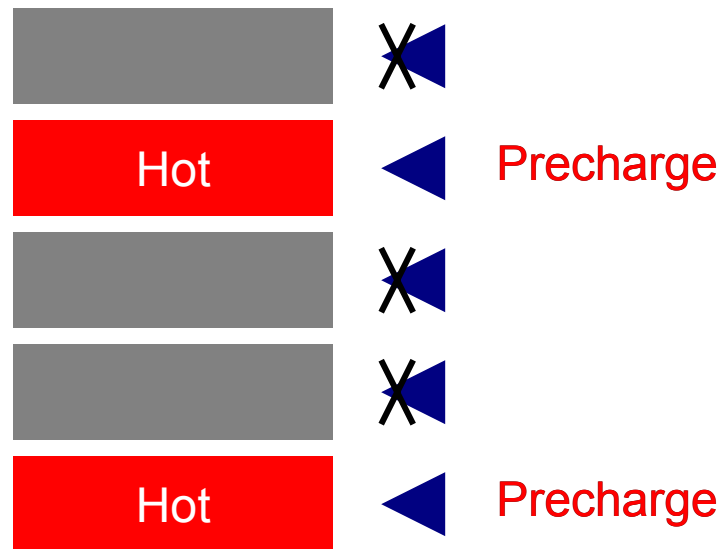
▶ Energy wasted in idle subarrays !



Exploit Temporal Locality in Subarrays

Observation

- ❑ All subarrays precharge/leak
- ❑ But, only small # of active subarrays



Contribution: Gated Precharging

Precharge only active subarrays

Detect temporal locality

- ❑ Decay counters
- ❑ Threshold comparison logic

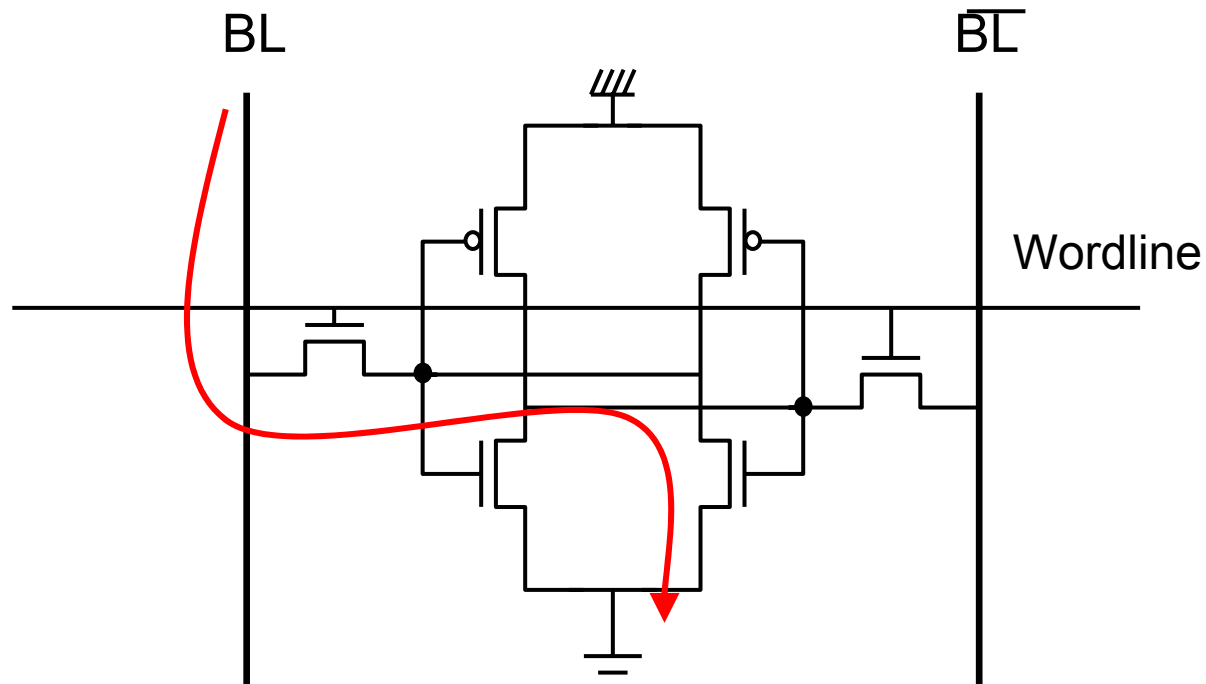
Reduce precharging

- ❑ by 89% in L1 d-cache
- ❑ by 92% in L1 i-cache
- ❑ with $< 2\%$ performance impact

Outline

- Overview
- ☛ Bitline Leakage
- Gated Precharging
 - Temporal Locality in Subarrays
 - Implementation
 - Gating Overhead
- Related Work
- Results
- Conclusion

Bitline Leakage in SRAM Cells



More than 60% discharge in 0.10μ

How Much Temporal Locality?



We evaluate, in a small window

1. How many accesses reuse subarrays?
2. How many active subarrays?

Subarray Reuse Ratio

Even in a small window, high subarray reuse

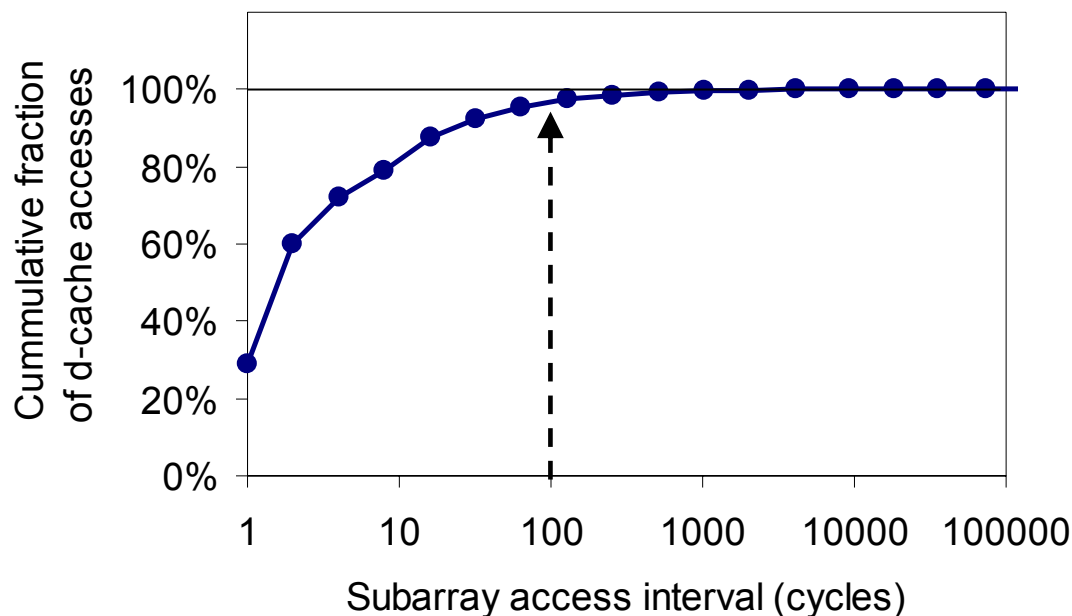
e.g., *gcc* with 32K L1D with 1K subarrays

❑ 96% accesses reuse subarrays in 100-cycle window

For all benchmarks,
in 100-cycle window

❑ 95% for d-cache

❑ 98% for i-cache



Fraction of Subarrays Accessed

In a small window, small # of active subarrays

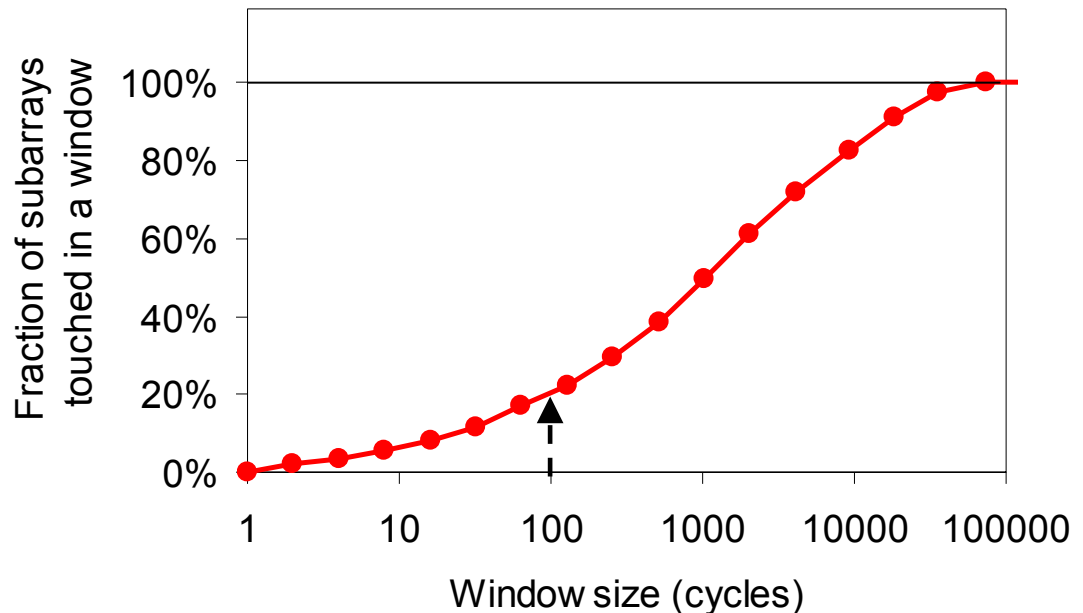
e.g., *gcc* with 32K L1D with 1K subarrays

❑ 19% of subarrays accessed in 100-cycle window

For all benchmarks,
in 100-cycle window

❑ < 29% for d-cache

❑ < 22% for i-cache



Temporal Locality in Subarrays



In 100-cycle window,

□ >95% of cache accesses reuse < 30% of subarrays

Most accesses temporally localized in small # of subarrays

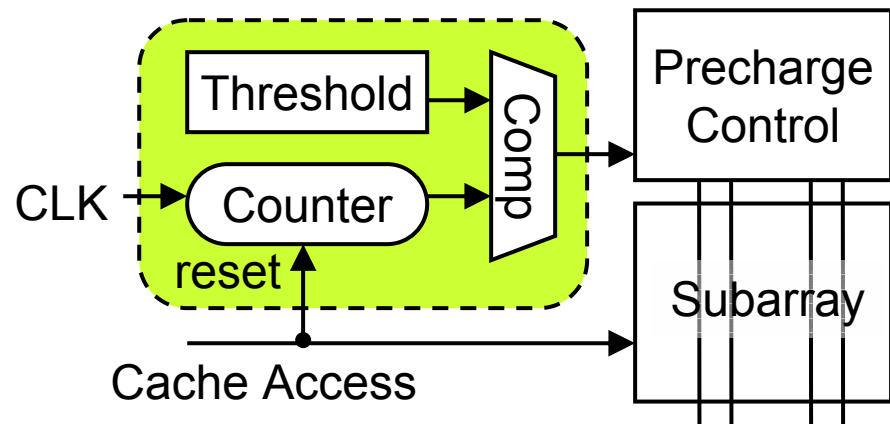
Gated Precharging: Hardware

Decay counter per subarray [Kaxiras, et al.]

Threshold value to decide “when” to precharge

Algorithm

- ❑ if count < threshold
⇒ precharge
- ❑ if count > threshold
⇒ no precharge



Gated Precharging: Overhead

Minimal performance overhead

- ❑ Hits on idle subarrays incur 1 extra cycle
- ❑ Infrequent due to temporal locality
(Example: *gcc* < 8% d-cache accesses)

Minimal energy overhead

- ❑ 10-bit counter per subarray
- ❑ Comparison logic
- ❑ Existing precharge control logic

Related Work



Delayed precharging [Alpha 21264]

- ❑ Precharge only required subarrays
- ❑ Increase cache access latency by delaying precharge

Resizable caches [Albonesi] [Yang, et al.]

- ❑ Capture working set size variation & resize caches
- ❑ Coarse switching granularity (time & space)
- ❑ Relatively larger performance overhead

Way prediction [Powell, et al., Inoue, et al.]

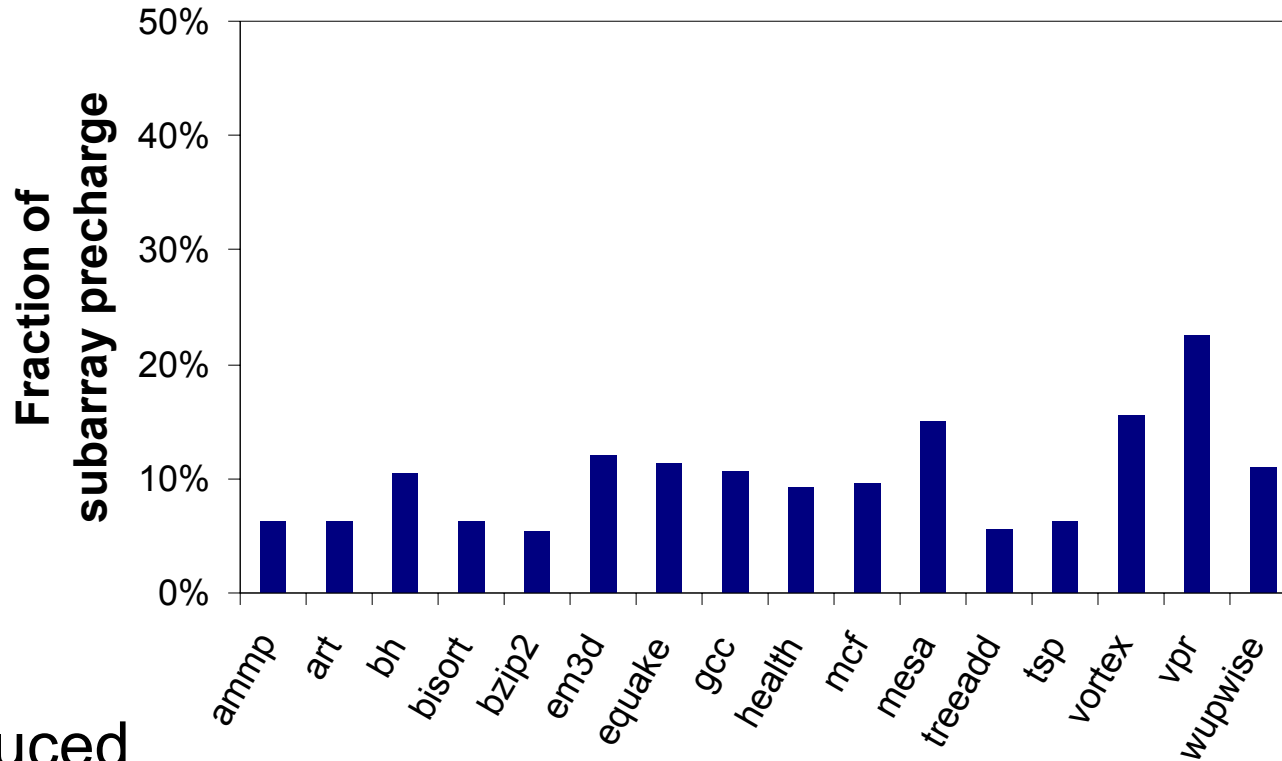
- ❑ Predict set associative way for next access
- ❑ Orthogonal to gated precharging

Methodology



- ❑ Wattach [ISCA2000]
 - 16 SPEC2000/Olden benchmarks
 - Performance impact < 2%
- ❑ Base Case
 - 8-wide issue, 128-entry RUU
 - 32K direct-mapped L1 I & D w/ 1K-subarray
 - 512K 4-way unified L2
- ❑ Determine threshold values based on profiling
 - Threshold values \cong 100 cycles

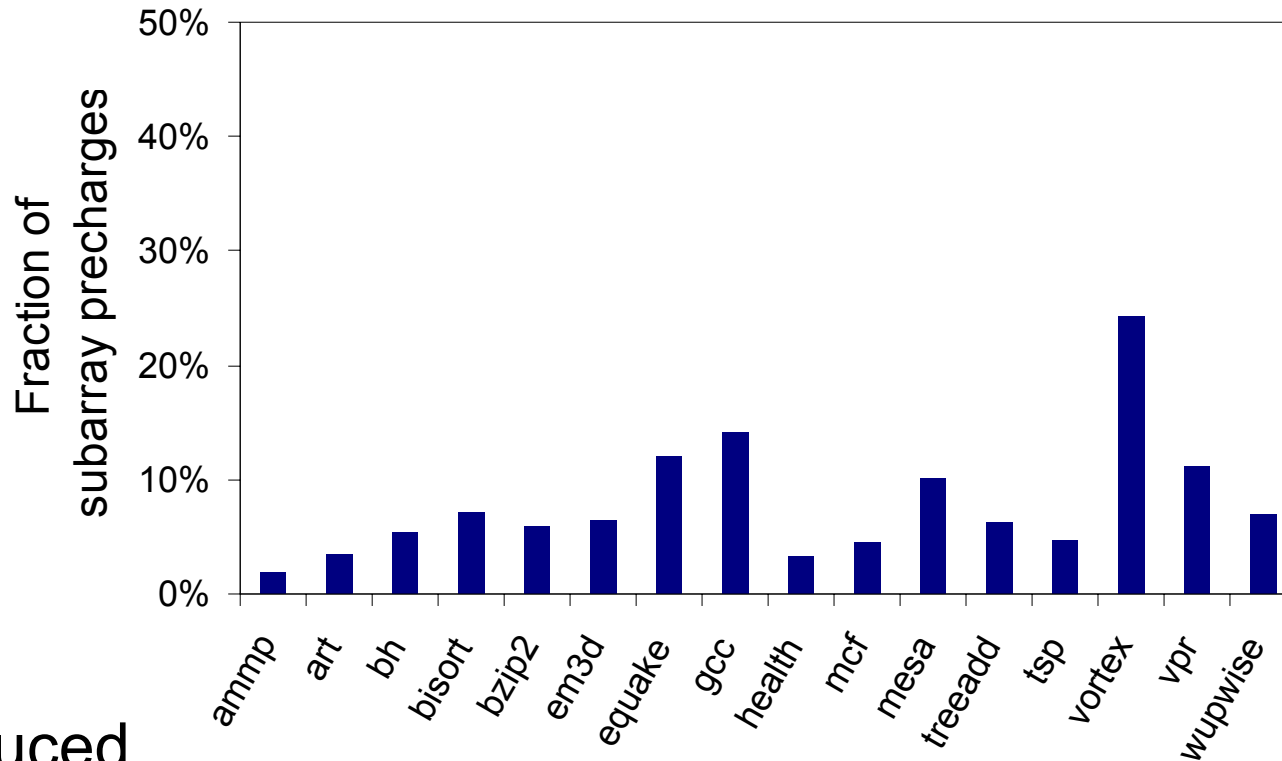
Results: D-Cache



Reduced

- ❑ On average by 89%
- ❑ by >85% for all but *vpr*

Results: I-Cache



Reduced

- ❑ On average by 92%
- ❑ by >90% for 13 benchmarks

Conclusions



High bitline leakage in deep submicron caches
Energy wasted in idle subarrays

Gated precharging

- ❑ Exploits temporal locality in subarrays
- ❑ Reduces 90% of precharging
- ❑ With $< 2\%$ performance impact

For more information



PowerTap Project

<http://www.ece.cmu.edu/~powertap>

Computer Architecture Lab

Carnegie Mellon University