Exploiting Page Table Locality for Agile TLB Prefetching

Georgios Vavouliotis[1,3], Lluc Alvarez[1,3], Vasileios Karakostas[4], Konstantinos Nikas[4], Nectarios Koziris[4], Daniel A. Jiménez[2], and Marc Casas[1,3]

[1]Barcelona Supercomputing Center [2]Texas A&M University [3]Universitat Politècnica de Catalunya [4]National Technical University of Athens

1. Virtual Memory

- Each memory access requires a virtual-to-physical address translation
- Modern systems implement virtual memory based on paging

Architectural Support for Paging-based Virtual Memory

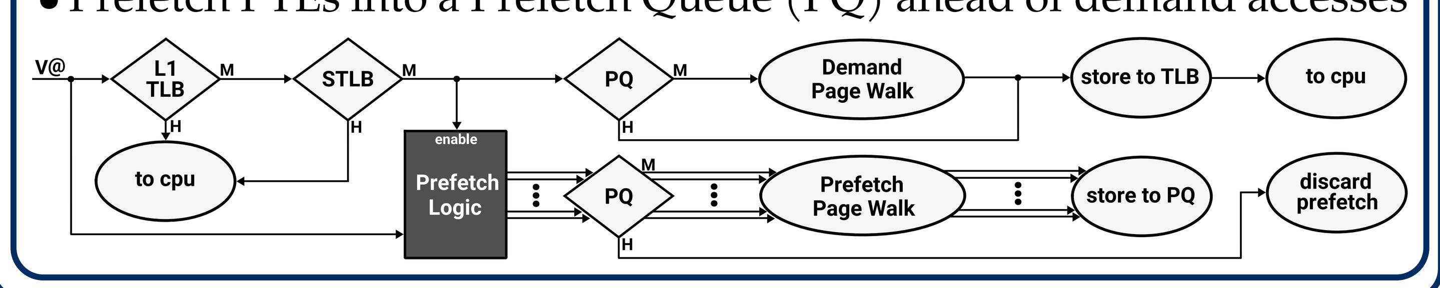
- **Page Table** stores virtual-to-physical mappings of all pages loaded to memory; x86 implements multi-level radix-tree Page Tables
- **TLBs** cache frequently used Page Table Entries (PTEs)
- **MMU Caches** store intermediate levels of the Page Table

3. Related Work

- SW/HW schemes that increase TLB reach [1]
- Approaches that accelerate TLB misses [2]
- Prefetching schemes that eliminate TLB misses [3]

Our Objective → TLB Prefetching

- Prefetch PTEs into a Prefetch Queue (PQ) ahead of demand accesses



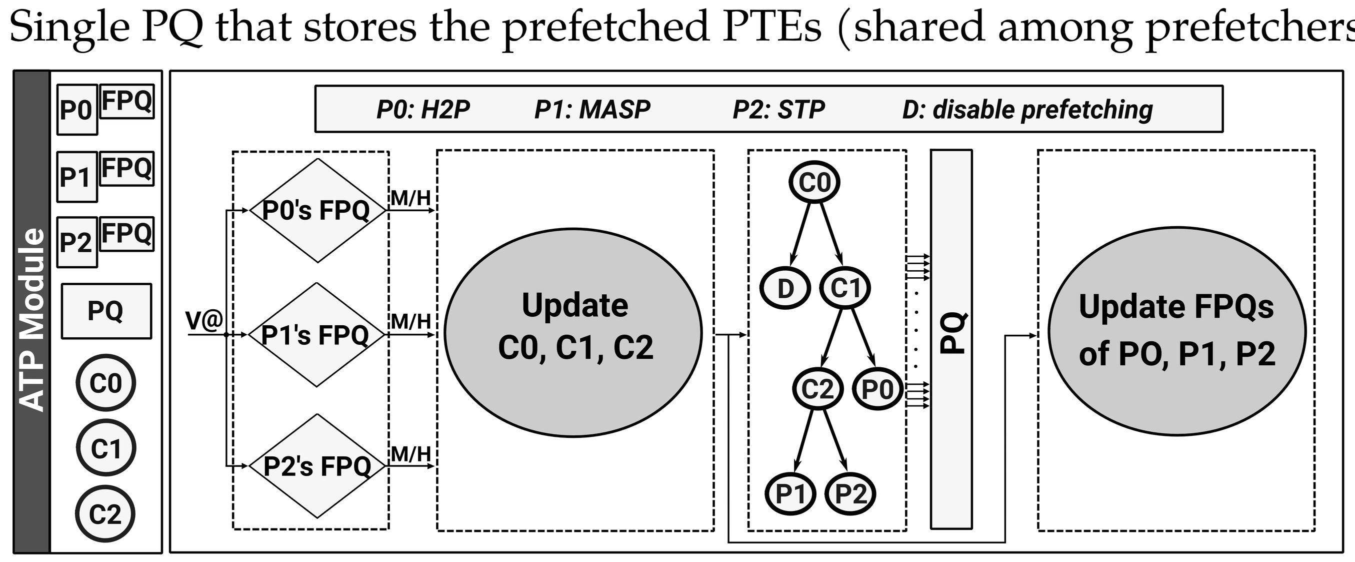
4. Agile TLB Prefetcher (ATP)

Analysis Findings

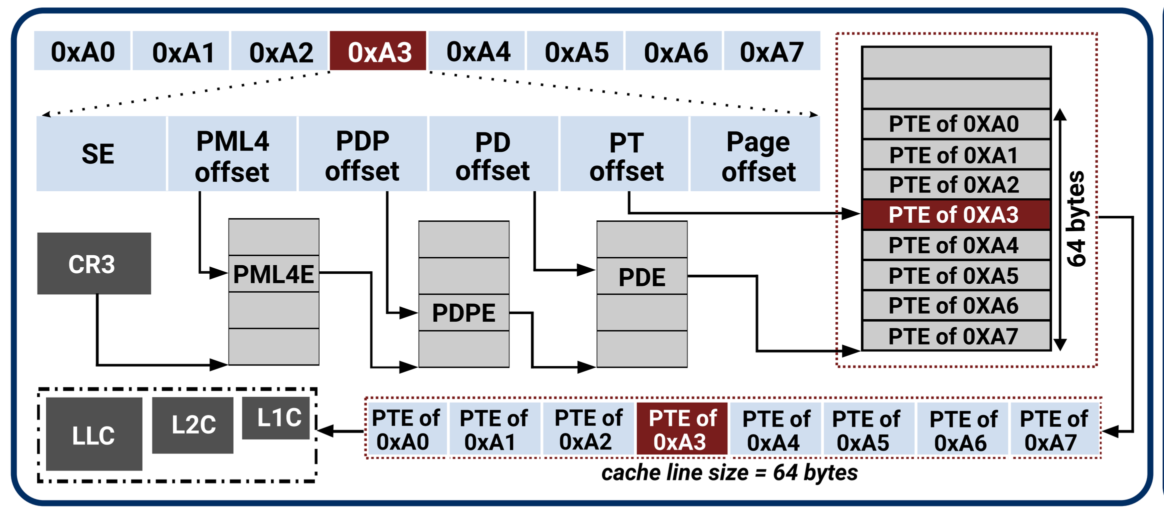
- Different TLB prefetchers perform best for different type of workloads
- TLB prefetching is not helpful during all execution phases

ATP's Design & Operation

- Combines three low-cost TLB prefetchers
  - Stride TLB Prefetcher (STP) → statically uses strides ±1, ±2
  - Modified Arbitrary Stride TLB Prefetcher (MASP) → table-based prefetcher that correlates patterns with the PC
  - H2 Prefetcher (H2P) → stateless distance-based prefetcher [3]
- One Fake Prefetch Queue (FPQ) per constituent prefetcher
- Selection and throttling logic implemented with saturating counters
- Single PQ that stores the prefetched PTEs (shared among prefetchers)



2. x86 Page Table Walking



Address Translation Bottleneck

- TLB misses cause long-latency page walks
- Frequent TLB misses deteriorate system's performance
- Increase in applications' working set sizes outpaces the increase in TLB sizes
- Workloads with massive data footprints exacerbate TLB pressure

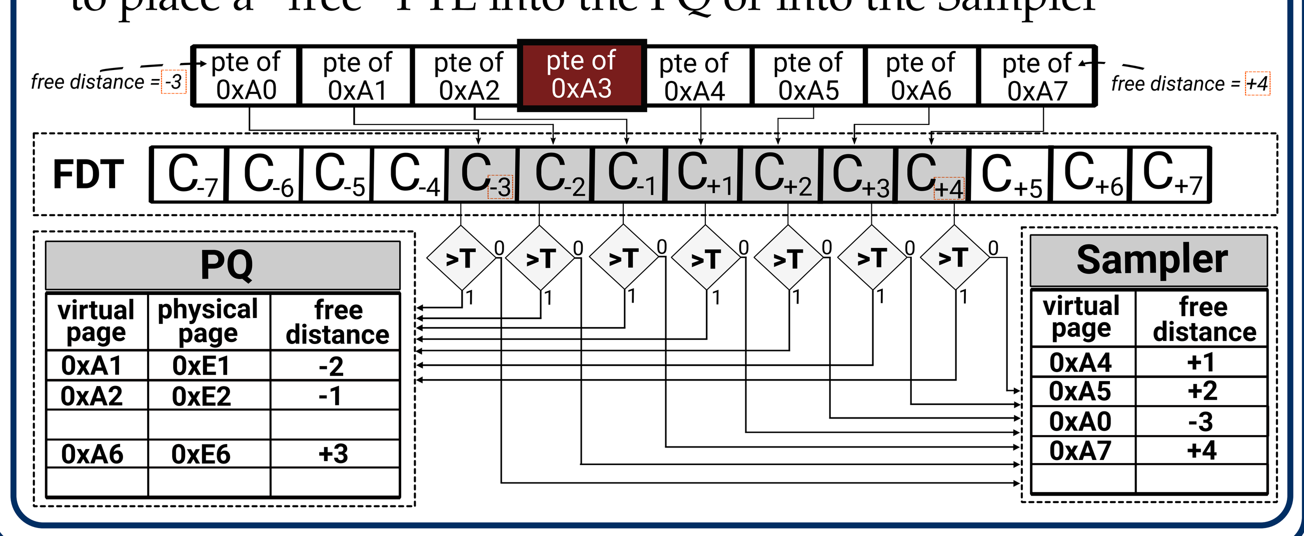
Page Table Locality & Free TLB Prefetching

- After a page walk the requested PTE coupled with 7 "free" PTEs are stored into a single cache line
- The cache-line adjacent PTEs can be prefetched for "free", without additional memory operations

5. Sampling-based Free TLB Prefetching (SBFP)

SBFP's Design & Operation

- **Free distance** → distance between the PTE that holds the demand translation and a "free" PTE within the cache line
  - Possible free distances: from -7 to +7, excluding zero
- **Sampler** → buffer that examines the usefulness of free distances which were useless in previous execution phases
- **PQ** → buffer that stores only the useful "free" PTEs per page walk (demand or prefetch)
- **Free Distance Table (FDT)** → table composed of 14 saturating counters (one per possible free distance) that decides whether to place a "free" PTE into the PQ or into the Sampler



Analysis Findings

- Exploiting page table locality for TLB prefetching has the potential to improve performance
- Exploiting page table locality for TLB prefetching reduces the page walk references to the memory hierarchy (L1C, L2C, LLC, DRAM)
- Prefetching all "free" PTEs per page walk provides suboptimal performance benefits

Key Properties

- SBFP can be combined with any TLB prefetching scheme without modifications
- SBFP can operate on both demand and prefetch page walks
- SBFP reduces the negative impact of prefetch page walks on memory references

6. Methodology

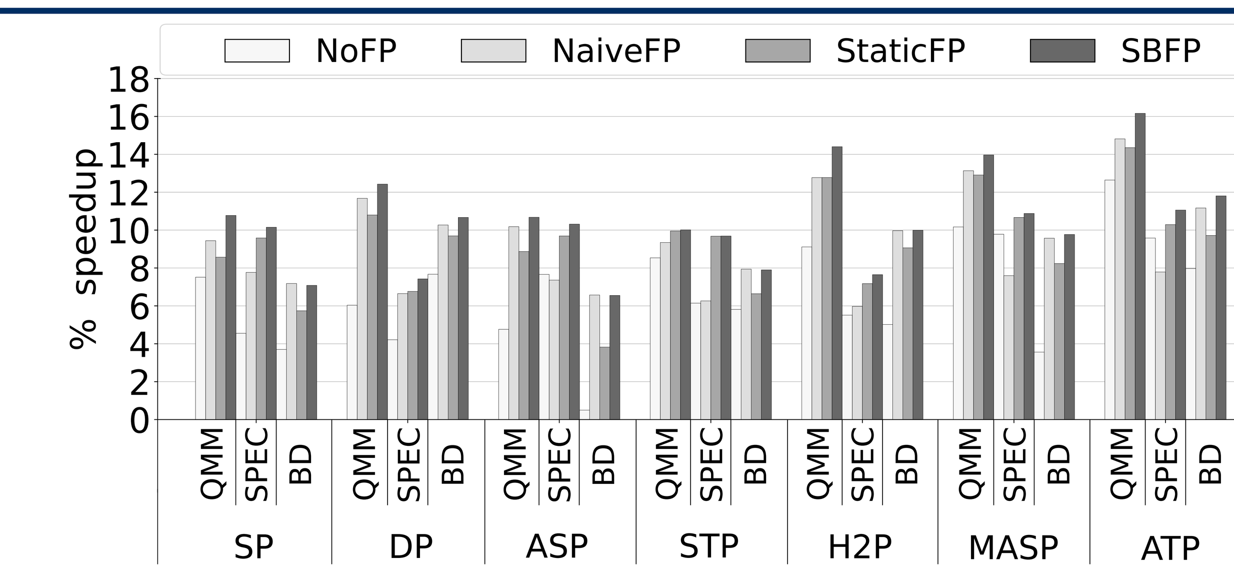
- ChampSim simulator (L2-TLB: 1536 entries, LLC: 2MB, DRAM: 4GB)
- The baseline system does not have prefetching at any TLB level
- Workloads: 12 SPEC CPU 2006 & 2017, 11 GAP, 2 XSBench, and 125 Qualcomm traces from the first Championship Value Prediction. We refer to GAP and XSBench workloads as Big Data (BD) workloads

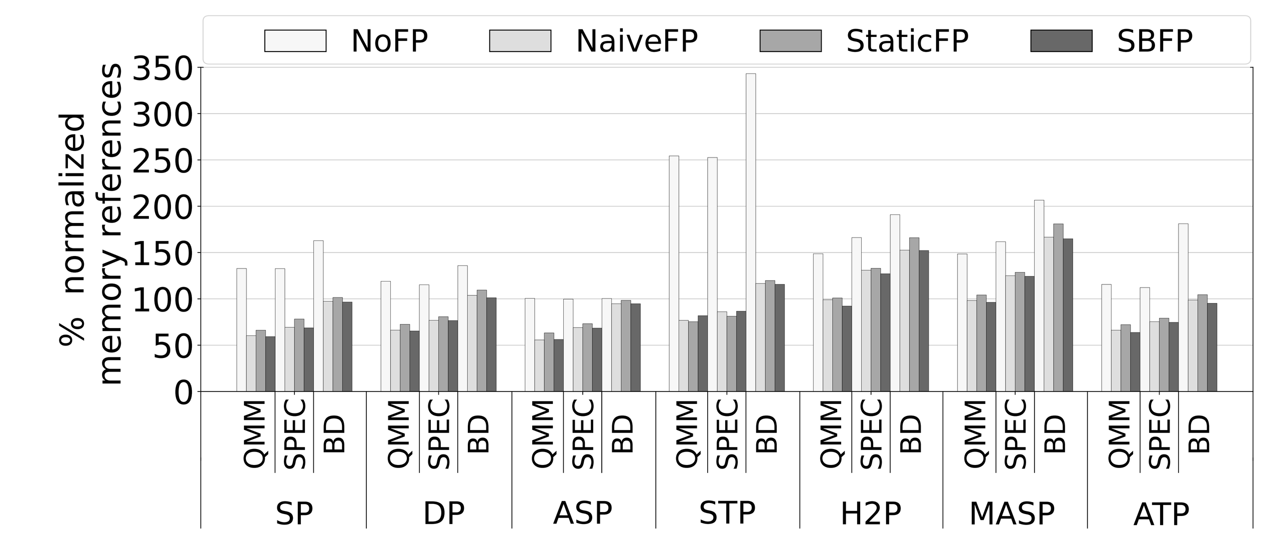
Evaluated TLB Prefetchers

- SP [3]
- DP [3]
- ASP [3]
- STP
- H2P
- MASP
- ATP

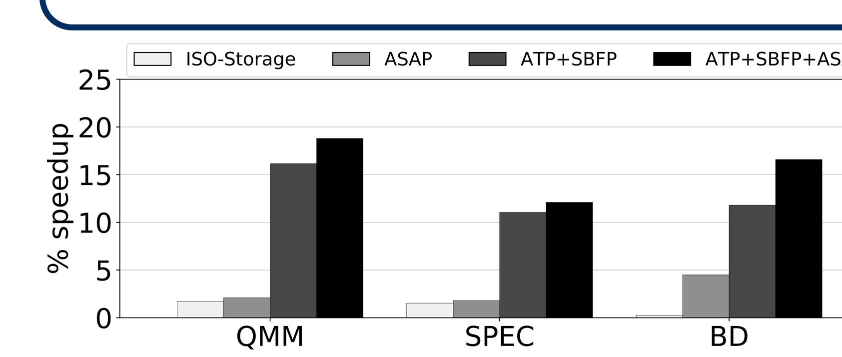
7. Evaluation

- Compare SBFP with scenarios that exploit page table locality
  - Free prefetching is not exploited (NoFP)
  - All "free" prefetches are placed in the PQ (NaiveFP)
  - Each prefetcher uses its own optimal set of free distances based on static offline exploration (StaticFP)





- NaiveFP, StaticFP, and SBFP experience higher performance gains and less memory references due to page walks than NoFP for all prefetchers
  - Free prefetching provides PQ hits that reduce demand page walks
  - Free prefetching saves costly prefetch page walks
- ATP+SBFP outperforms the best prior TLB prefetcher by 8.7%, 3.4%, and 4.2% for the QMM, SPEC, and BD workloads, respectively
- ATP with SBFP eliminates by 37%, 26%, and 5% the page walk memory references for the QMM, SPEC, and BD workloads, respectively



- ATP+SBFP outperforms the ISO-Storage scenario
- ASAP [4] improves the performance of ATP+SBFP

[1] Pham et al., "CoLT: Coalesced Large-Reach TLBs", HPCA'12
[2] Bhattacharjee, "Large-reach memory management unit caches", MICRO'13
[3] Kandiraju et al., "Going the Distance for TLB Prefetching: An Application-driven Study", ISCA'02
[4] Margaritov et al., "Prefetched Address Translation", MICRO'19

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 955606.

8. Follow-up Work

Morrigan: A Composite Instruction TLB Prefetcher – MICRO'21

Instruction TLB prefetching for big code applications