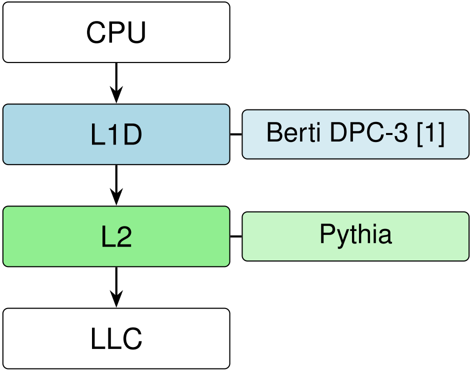
Pushing the Limits of the Berti Prefetcher

Simranjit Singh[1], Agustín Navarro-Torres[2], Alberto Ros[1]

[1]University of Murcia (simranjit.singh@um.es, aros@ditec.um.es)
[2]Universidad de Zaragoza (agusnt@unizar.es)

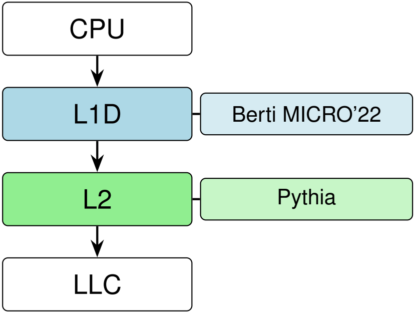
Feb 1, 2026

BASELINE



[1] DPC-3 = 3rd Data Prefetching Championship (2019)

BASELINE: BERTI DPC-3 → MICRO'22

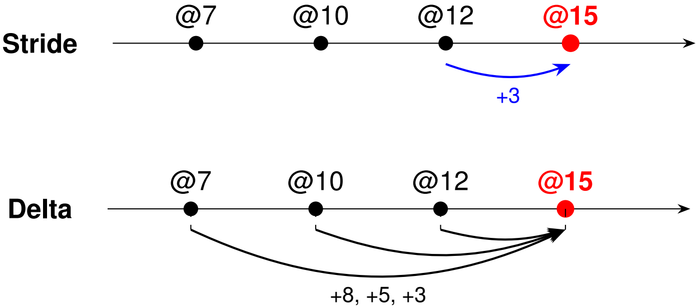


MICRO'22 Improvements

- Simplified design
- Virtual addresses
  (enables cross-page prefetching)
- Local per-IP deltas
  (vs memory regions)
- Timeliness via latency tracking
- Confidence-based delta selection

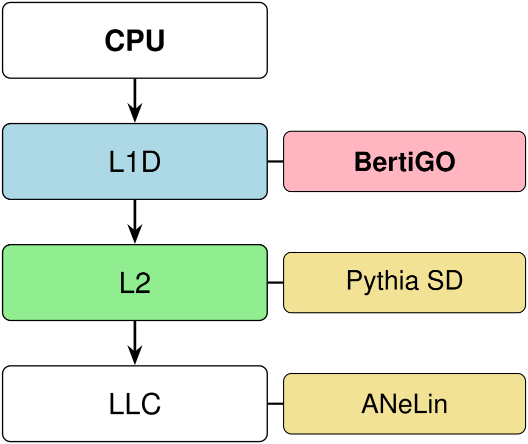
BERTI MICRO'22: KEY IDEAS

Stride vs Local Deltas

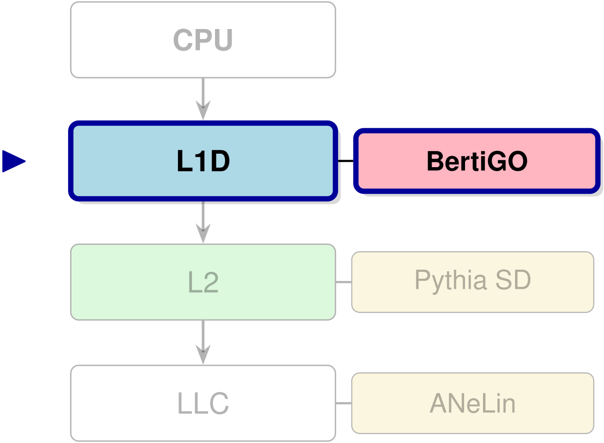


Per-IP deltas • Latency tracking • Confidence-based L1D/L2 • Virtual addresses

Our Approach



Our Approach



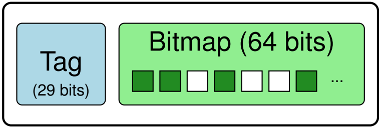
Inefficiency in Berti: Redundant Requests

Berti prefetches lines
already in cache

→ Uses PQ slots
→ Uses port slots

Solution 1: Region-Based Bitmap Filter

Filter Entry



Tag
(29 bits)

Bitmap (64 bits)
■ ■ □ ■ □ □ ■ ...

How it works:

1. On access/prefetch: **Set bit**
2. Before prefetch: **Check bit**
3. Bit set? → **Skip prefetch!**
4. On L1D eviction: **Clear bit**

Learns Useless Prefetches

If L2 prefetch never promoted to L1D:

→ No eviction
→ Bit stays set
→ Future requests blocked

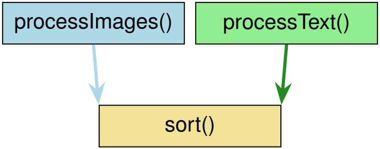
Storage: 15.6 KB
1360 entries · NRU
Tracks 87,040 lines

Inefficiency in Berti: Missing Context

Berti indexes by IP alone

Can we add new input sources?

Solution 2: Context-Aware Prefetching



Same IP, different patterns

Highly effective for AI/ML

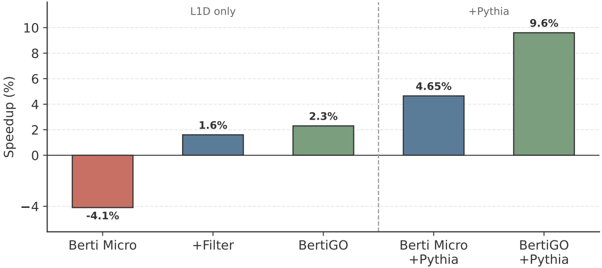
Llama (full BW): **+20-50%** vs IP-only

Storage: <0.01 KB

How it works:

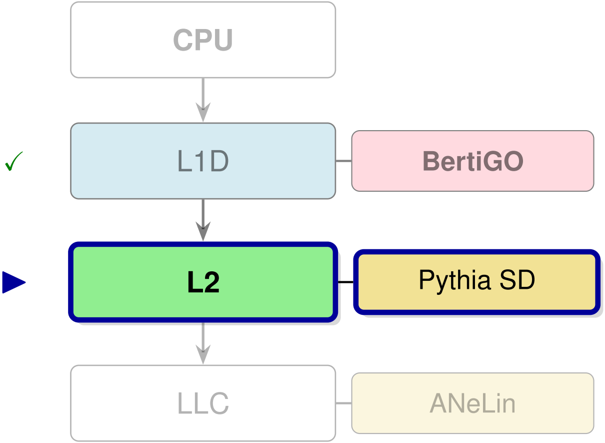
1. Track last 4 IPs
2. Hash into path signature
3. Query deltas using both IP alone and path signature
4. Merge predictions

BERTIGO: L1D CONTRIBUTION

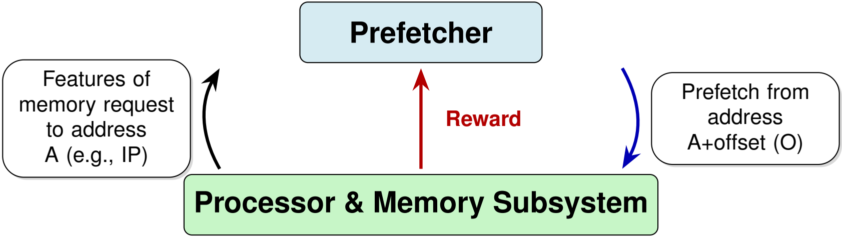


Full BW, over baseline

Our Approach



Pythia: Key Ideas



Prefetcher

Features of memory request to address A (e.g., IP)

Reward

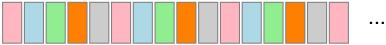
Prefetch from address A+offset (O)

Processor & Memory Subsystem

RL Learning • Multiple program context features • Bandwidth-aware rewards

L2: SET-DUELING FOR PYTHIA

2048 Cache Sets



🟥 NoPref 🟦 IP 🟩 IP_Delta

🟧 IP∨IP_Delta ■ IP∧IP_Delta

How it works:

1. Each set assigned to one candidate
2. Track miss rate (misses ÷ accesses)
3. Winner if ≥4% better miss rate than NoPrefetch

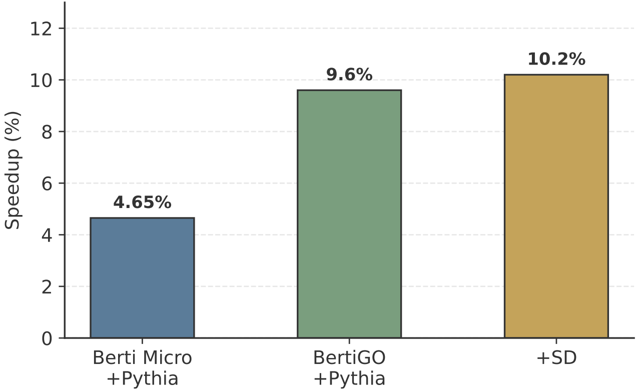
Workload-Driven

Default policy not always optimal

→ Let workload decide

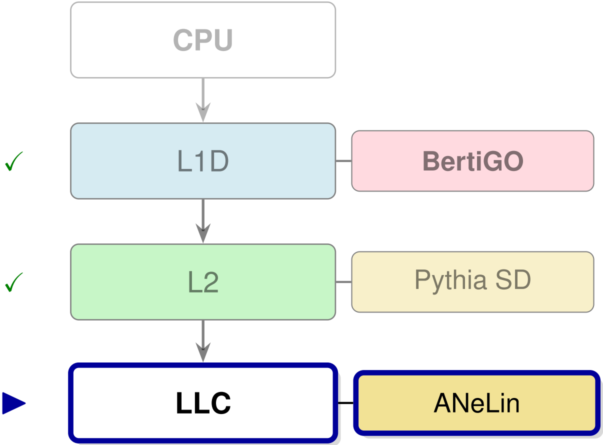
Storage: 90 bytes

Results: BertiGO + Pythia SD

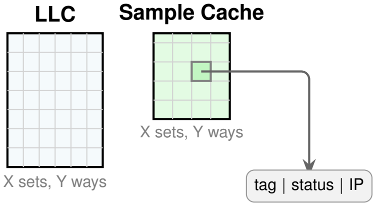


Full BW, over baseline

Our Approach



LLC: ANeLin (Adaptive Next-Line)



How it works:

1. On access: track next-line in sample cache; prefetch if enabled
2. Demand hit → timely[1] or late[2]
3. Unused eviction → useless
4. Saturate: enable next-line if useful $\gg$ useless; decay counters 75%

Two-Level Filtering

Per-Core: enable for workload?

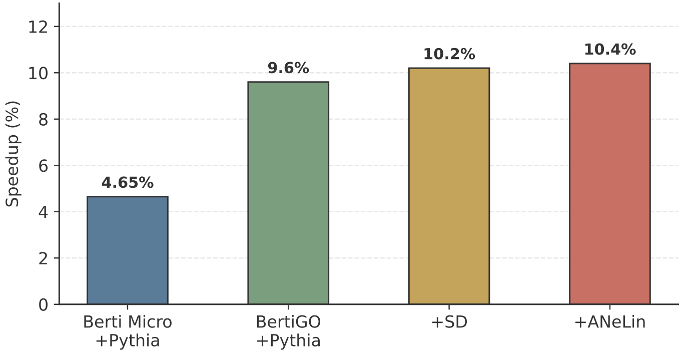
Per-IP: enable for this IP?

Storage: 209 KB

[1] Timely: $>$ mean lat → 1 useful

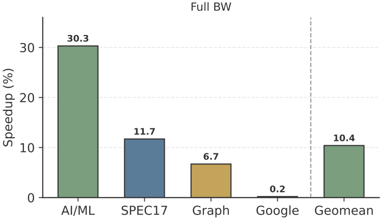
[2] Late: $>$ 0.2× mean lat → 0.5 useful

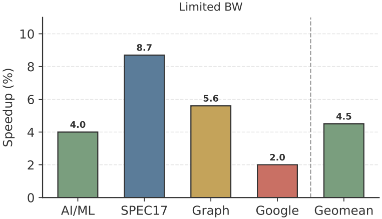
Results: Full Configuration



Full BW, over baseline

Evaluation


Full BW
Speedup (%)
0
10
20
30
30.3
11.7
6.7
0.2
10.4
AI/ML
SPEC17
Graph
Google
Geomean


Limited BW
Speedup (%)
0
2
4
6
8
10
4.0
8.7
5.6
2.0
4.5
AI/ML
SPEC17
Graph
Google
Geomean

Conclusion

L1D: BertiGO

Region Filter

- Eliminates redundant requests
- Learns useless L2 prefetches

IP-Path Signatures

- Context-aware predictions
- Effective for AI/ML

L2: Pythia SD

Set-Dueling

- First applied to prefetcher feature selection
- Choose optimal policy per workload

LLC: ANeLin

Adaptive Next-Line

- Per-IP and per-core learning

Thank You

Questions?

L1D: BertiGO

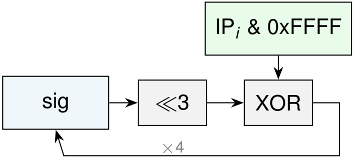
L2: Pythia Set-Dueling

LLC: ANeLin

Simranjit Singh, Agustín Navarro-Torres, Alberto Ros — DPC-4

Backup: IP-Path Signature

```
sig ← 0
for i = 3 downto 0:
    sig ← (sig ≪ 3) ⊕
          ($IP_i$ & 0xFFFF)
return sig
```



Context A

0x12, 0x34, 0x56, 0x78, 0xAB

⇓

0x15EEB

Context B

0xFF, 0x11, 0x22, 0x33, 0xAB

⇓

0xFDBB3