Constable: Improving Performance and Power Efficiency by Safely Eliminating Load Instruction Execution

*Rahul Bera[1] *Adithya Ranganathan[2] Joydeep Rakshit[2] Sujit Mahto[2]
Anant V. Nori[2] Jayesh Gaur[2] Ataberk Olgun[1] Konstantinos Knellopoulos[1]
Mohammad Sadrosadati[1] Sreenivas Subramoney[2] Onur Mutlu[1]

[1] ETH zürich [2] intel

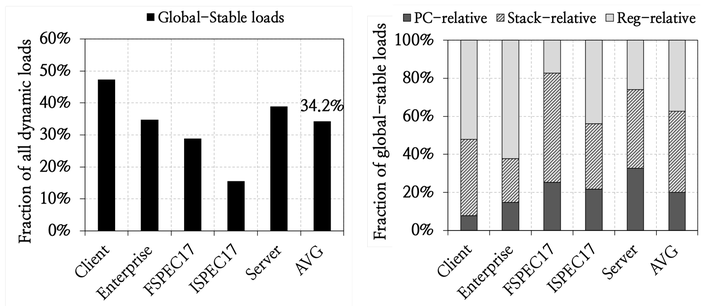
arXiv

Load Inspector

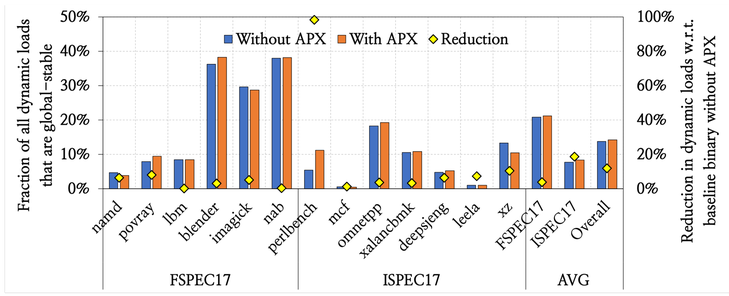
1. Key Problem

- Load instructions limit instruction-level parallelism (ILP) due data and resource dependence.
- Even though techniques like load value prediction (LVP) and memory renaming (MRN) mitigate load data dependence, they fail to mitigate resource dependence.

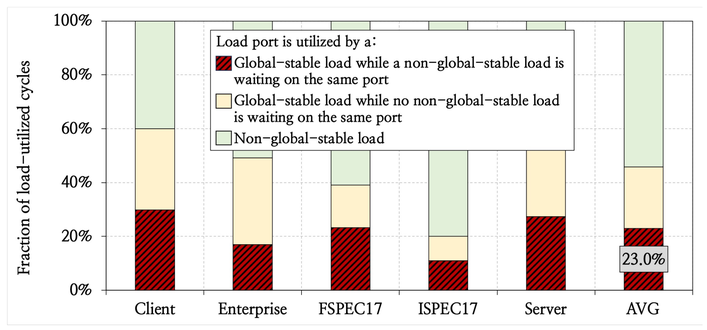
2: Key Observations



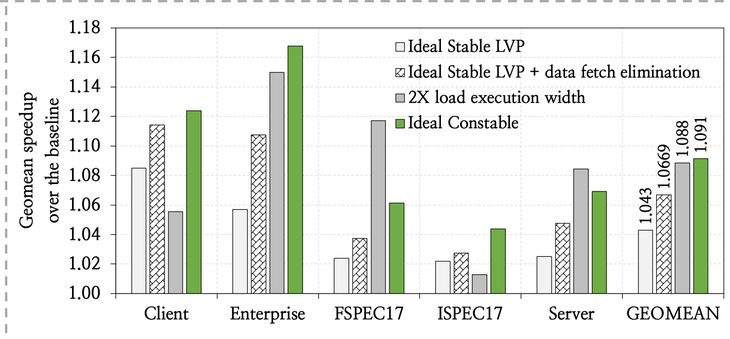
~1 in every 3 dynamic loads fetch same data from same address across the entire workload



Doubling x64 registers has negligible impact on global-stable loads at compile time



Global-stable loads cause significant resource dependence even in presence of LVP & MRN



Mitigating resource dependence has high performance headroom

3. Our Goal & Proposal

To improve instruction level parallelism by mitigating both load data and load resource dependence

CONSTABLE

A purely-microarchitectural technique that mitigates both load data and load dependence by safely eliminating the entire execution of certain load instructions

4. Constable: Key Insight & Idea

Two successive dynamic instances of a load instruction will fetch *the same data from the same address* if

- None of their source registers has changed, and
- No store/snoop request has arrived at their address

Dynamically identifies load instructions that have historically fetched the same data from the same load address (i.e., *likely-stable*)

Eliminates execution of likely-stable loads by tracking modifications to their source registers and their load addresses

5. Constable Design

Stable Load Detector (SLD)

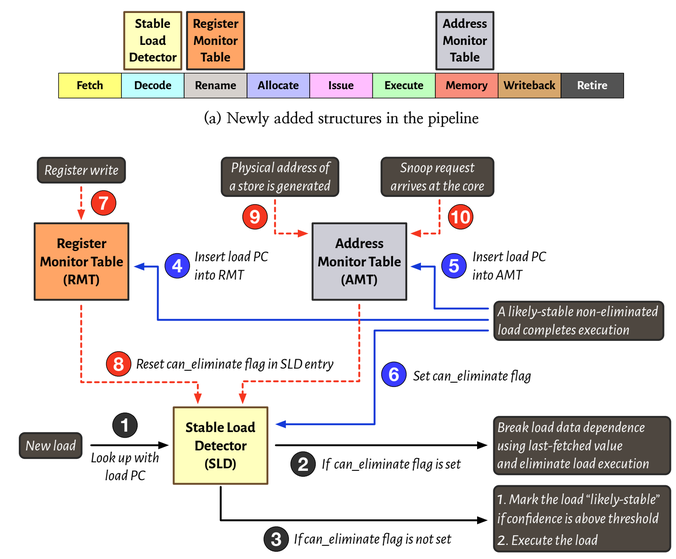
- Identifies likely-stable loads
- Decides whether to eliminate a load
- Provides last-executed load address and value

Register Monitor Table (RMT)

- Monitors modifications to arch. registers
- Stops elimination if a register gets modified

Address Monitor Table (AMT)

- Monitors modifications to physical address
- Stops elimination if an address gets modified



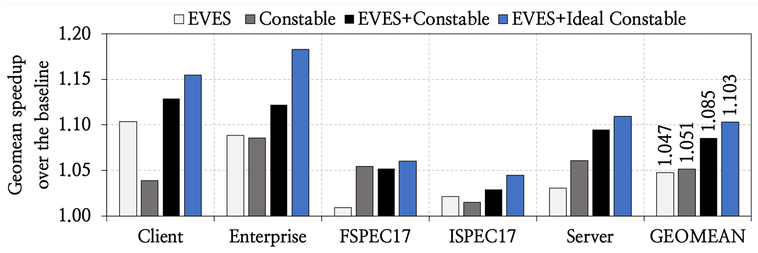
(a) Newly added structures in the pipeline

(b) Key operations in Constable

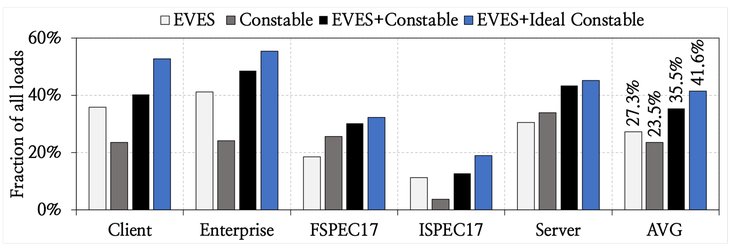
6: Evaluation & Key Takeaways

Methodology

- In-house industry-grade simulator
  - Aggressive baseline with memory renaming and dynamic instruction elimination techniques
- 90 workloads
  - All from SPEC CPU2017, Client, Enterprise, and Server applications
- Three comparison points
  - EVES [Seznec, CVP'18], ELAR [Bekerman+, ISCA'00], RFP [Shukla+, ISCA'22]
- Two configurations
  - noSMT & 2-way SMT



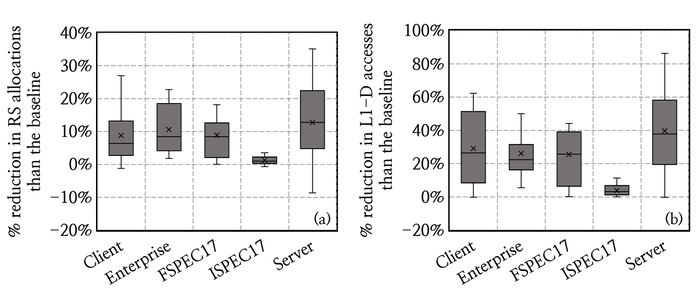
Performance benefit in noSMT configuration



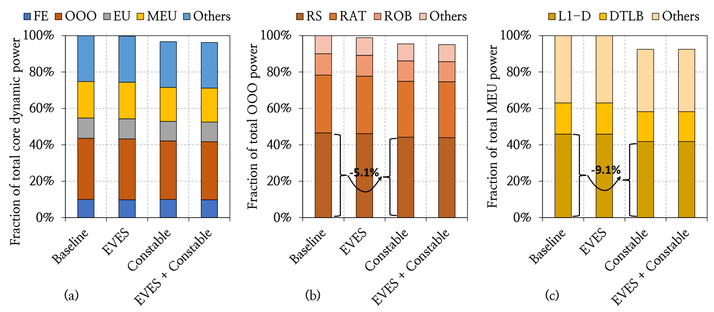
Elimination coverage of Constable

Key Results

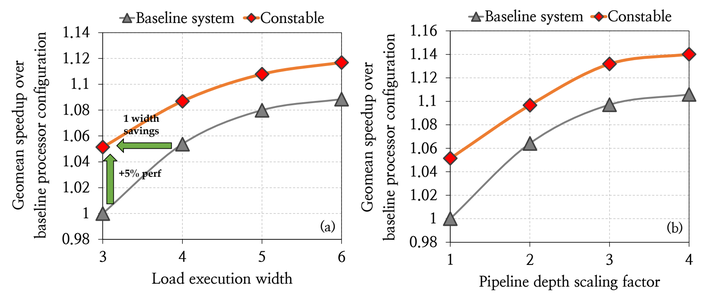
- **5.1% (up to 31.2%) and 8.8% (37.9%) performance** benefit in noSMT and 2-way SMT configurations
- **23.5%** of all dynamic loads are eliminated
- **8.8% and 26% reduction** in reservation station allocations and L1-data cache access
- **3.4% (up to 24.6%) reduction in total core power** consumption
- Only **12.4 KB/core** storage overhead



Reduction in pipeline resource utilization



Reduction in dynamic power consumption



Performance sensitivity with load execution width and pipeline depth