AVPP:

Address-first Value-next Predictor with Value Prefetching for Improving the Efficiency of Load Value Prediction

Lois Orosa, Rodolfo Azevedo and Onur Mutlu

SAFARI ETH Zürich

INSTITUTO DE COMPUTAÇÃO
Universidade Estadual de Campinas

Executive Summary

Motivation: single-thread performance is critical for many applications

Problem: improving single-thread performance in modern processors requires techniques that significantly increase the hardware cost

Goal: revisit Load Value Prediction as an efficient way to improve single-thread performance

Contributions:

- We propose optimizations for **reducing the hardware cost of load value prediction**
- We propose a **new taxonomy** of Value Prediction Policies
- **AVPP**:
  - New load value predictor that predicts the **address first** and the **value next.**
  - Increases the coverage of the predictor by **prefetching the value** of a future instance of the load instruction

Results:

- AVPP outperforms all **state-of-the-art** value predictors in the context of load value prediction, and it is **less complex** than predicting all instructions
- AVPP provides **11.2%** system performance improvement and **3.7%** energy savings compared to no value prediction.

AVPP Outline

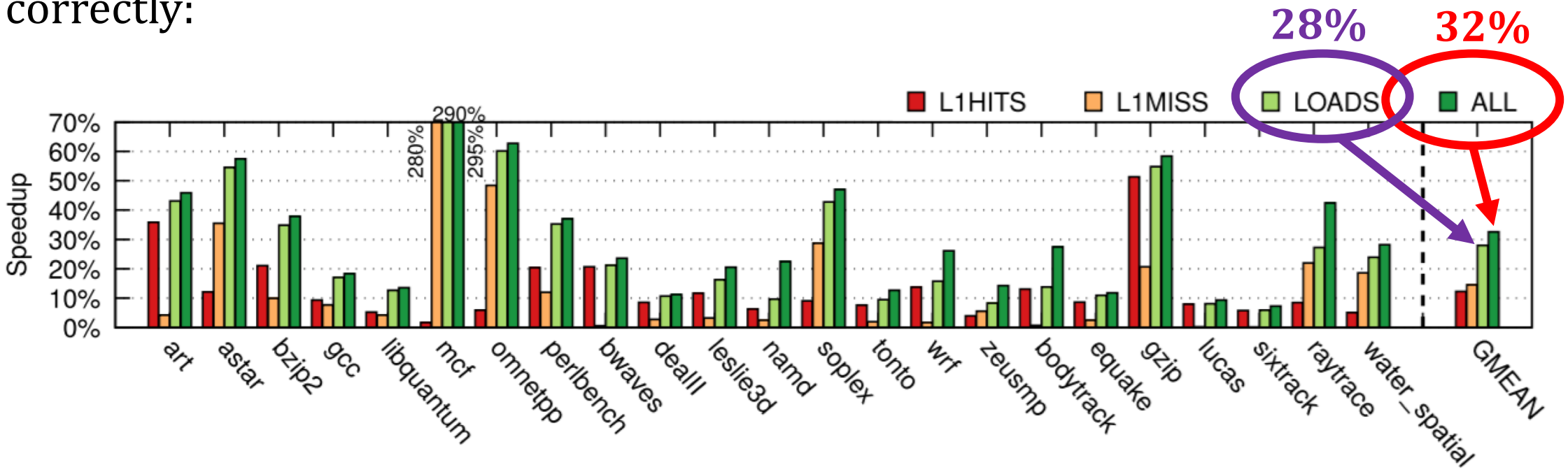
- Motivation and Goal
- Background
- Reducing Hardware Complexity
- Taxonomy of Value Prediction
- Mechanism: AVPP
- Evaluation and Results
- Conclusion

AVPP Outline

- Motivation and Goal
- Background
- Reducing Hardware Complexity
- Taxonomy of Value Prediction
- Mechanism: AVPP
- Evaluation and Results
- Conclusion

Motivation and Goal

- **True data dependencies limit single-thread performance significantly**
- We simulate an **oracle predictor** that predicts the output of instructions correctly:



- We observe that an oracle predictor achieves almost the same performance predicting load instructions as predicting all instructions
- Predicting all instructions requires large predictors
- **Our goal** is **revisit Load Value Prediction** as a **low-cost** alternative to improve single-thread performance

AVPP Outline

- Motivation and Goal
- Background
- Reducing Hardware Complexity
- Taxonomy of Value Prediction
- Mechanism: AVPP
- Evaluation and Results
- Conclusion

Background: Value Prediction

True Data Dependency:

instr_0: A = B + 1

instr_1: C = A + 3

Read-After-Write (RAW)

Depends on the result of the previous instruction

- **Value Prediction** breaks true data dependencies by predicting values and performing speculative execution
  - Prediction: the output value of instr_0 (A)
  - Speculation: executes instr_1 using the predicted value of A
  - instr_0 and instr_1 are executed in parallel

- Requires a Rollback mechanism to recover from mispredictions

State-of-the-art Value Predictors

- **Last Value Predictor (LVP)** [Lipasti+ MICRO'96]
  - The predicted value is the last value
- **Stride Predictor (Stride)** [Mendelson+ TR'96]
  - The predicted value is the last value + stride
  - **2D-Stride** [Eickemeyer+ IBM Journal'93]: variant that improves performance in loops
- **Finite Context Method (FCM) predictor** [Sazeides+ TR'97]
  - Context based predictor
  - Large prediction tables
  - **D-FCM** [Goeman+ HPCA'01]: variant that uses strides
- **VTAGE** [Perais+ HPCA'14]
  - Uses global branch history
  - **D-VTAGE** [Perais+ HPCA'15]: variant that uses strides

AVPP Outline

- Motivation and Goal
- Background
- **Reducing Hardware Complexity**
- Taxonomy of Value Prediction
- Mechanism: AVPP
- Evaluation and Results
- Conclusion

Reducing Hardware Complexity

- Predicting only load instructions instead of predicting all instructions has some advantages:
  - Decrease the hardware cost: we show that load value predictors have a smaller area footprint
  - Less pressure over shared resources (e.g., Physical Register File): we show how to reduce the number of ports in the paper
  - We leverage the Load Queue for implementing load value prediction

Load Instructions are 25% of all instructions on average

Small modifications to support Load Value Prediction

Previous Works on Reducing Complexity of Value Prediction

Value prediction for all instructions:

Work	[Perais+ HPCA'14]	EOLE [Perais+ ISCA'14]	BeBOP [Perais+ HPCA'15]	[Perais+ MICRO'16]
Predictor Entries	8192	8192	2048	2048
Speedup	~10%	~10%	11.2%	5%
Main contribution for reducing complexity	- Introduces Confidence Counters - Simple mechanism to recover from value mispredictions	- Reduces the extra ports required on the PRF - Increases the complexity of the in-order front-end and back-end	Single predictor entry for a cache line	- Register reuse - Low hardware cost

Load Value Prediction:

AVPP
512
11.2%
Minimal hardware Modifications to an OoO processor

AVPP Outline

- Motivation and Goal
- Background
- Reducing Hardware Complexity
- Taxonomy of Value Prediction
- Mechanism: AVPP
- Evaluation and Results
- Conclusion

Taxonomy of Value Prediction Policies

We propose a **new taxonomy** for analyzing value prediction design choices

1. Predictor update
2. Prediction availability
3. In-flight pending updates

Taxonomy: Predictor Update Policies

Where is the predictor **updated** in the pipeline?

1. **Correct update**:
   - The predictor is updated at the commit stage
2. **Speculative update**:
   - The predictor is updated at the fetch stage

Taxonomy: Prediction Availability Policies

What happens when the prediction is **not ready at dispatch time?**

1. **Delay dispatch**
   - The pipeline is stalled waiting for the prediction
2. **Not-delay dispatch**
   - The prediction is discarded

Taxonomy: In-flight Pending Update Policies

What happens with **back-to-back predictions?**

1. **In-flight ignore**
   - The predictor ignores previous in-flight instructions
2. **In-flight wait**
   - The predictor waits for previous in-flight instructions to update the predictor

Taxonomy

We evaluate different combinations of these policies for all predictors we evaluate

AVPP Outline

- Motivation and Goal
- Background
- Reducing Hardware Complexity
- Taxonomy of Value Prediction
- **Mechanism: AVPP**
- Evaluation and Results
- Conclusion

AVPP: Address-first Value-next Value Predictor with Data Prefetching

Observations:

- Predicting load instructions has almost the same potential performance benefits as predicting all instructions
- Addresses are usually more predictable than values

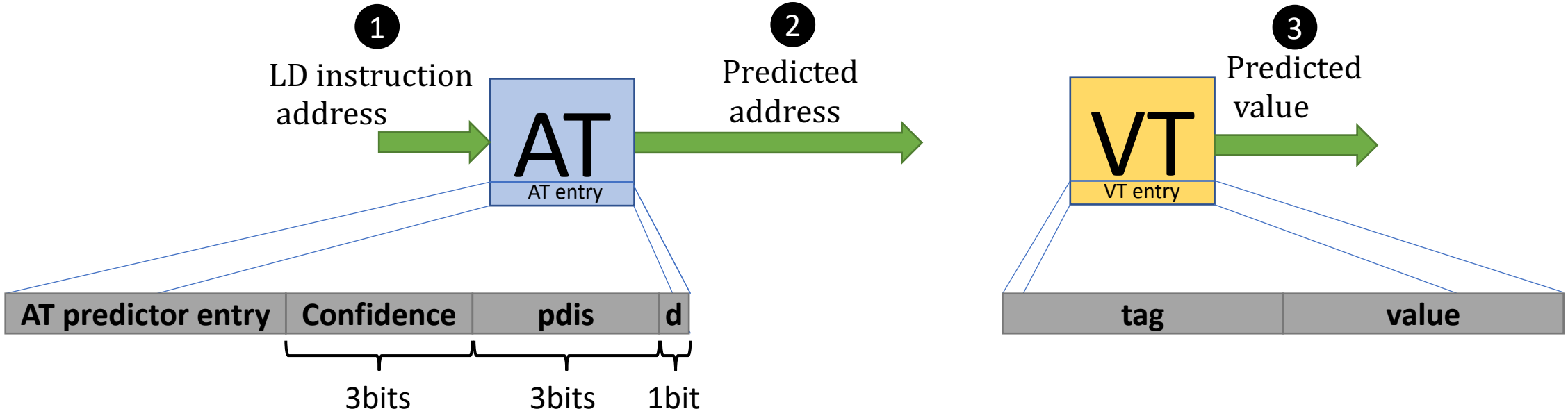
Key Ideas:

- Predict only load instructions to reduce hardware cost
- Leverage the better predictability of addresses on load instructions to improve performance

AVPP: Predictor Overview

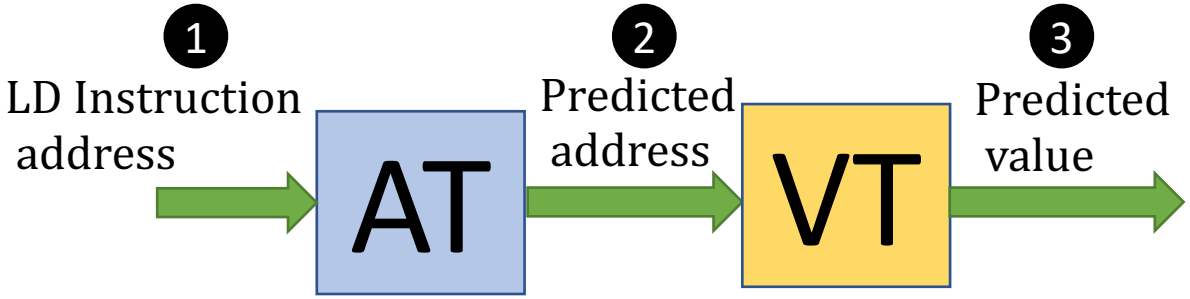
Two main hardware structures:

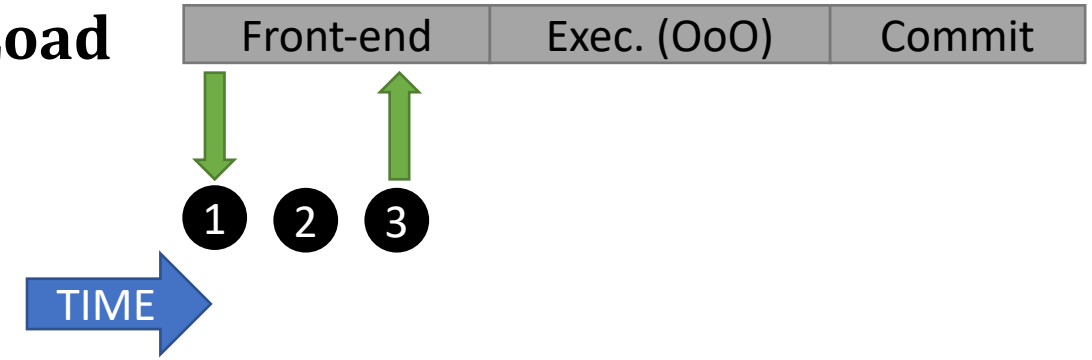
1. **Address Table (AT):**
   1. Implemented with any predictor
   2. Indexed by the load instruction address
   3. Calculates the predicted address
2. **Value Table (VT):**
   1. Indexed by the predicted address
   2. Returns the predicted value



AVPP: Prediction Overview (II)

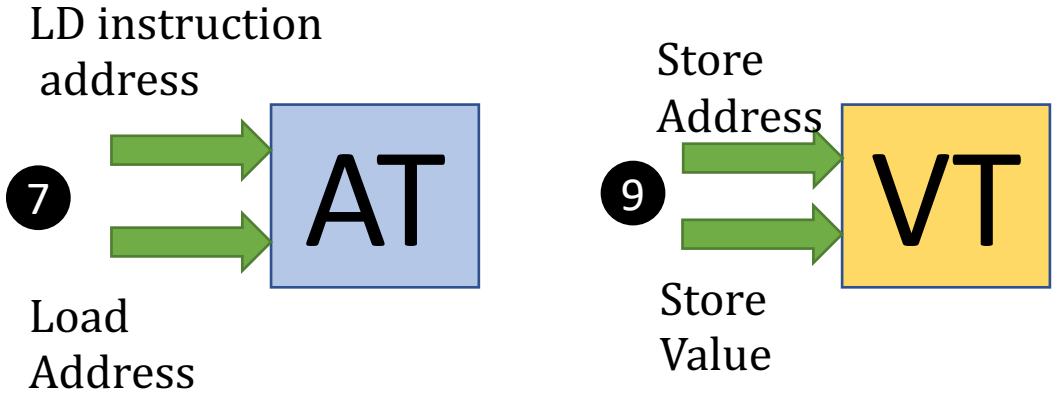
- AVPP predicts the output of every load instruction
- The prediction happens in the front-end of the pipeline ❶❷❸
- Confidence mechanism to improve accuracy [Perais+ HPCA'14]
- If the prediction is confident, write the result in the target register ❸
- Speculatively execute all the dependent instructions ❹

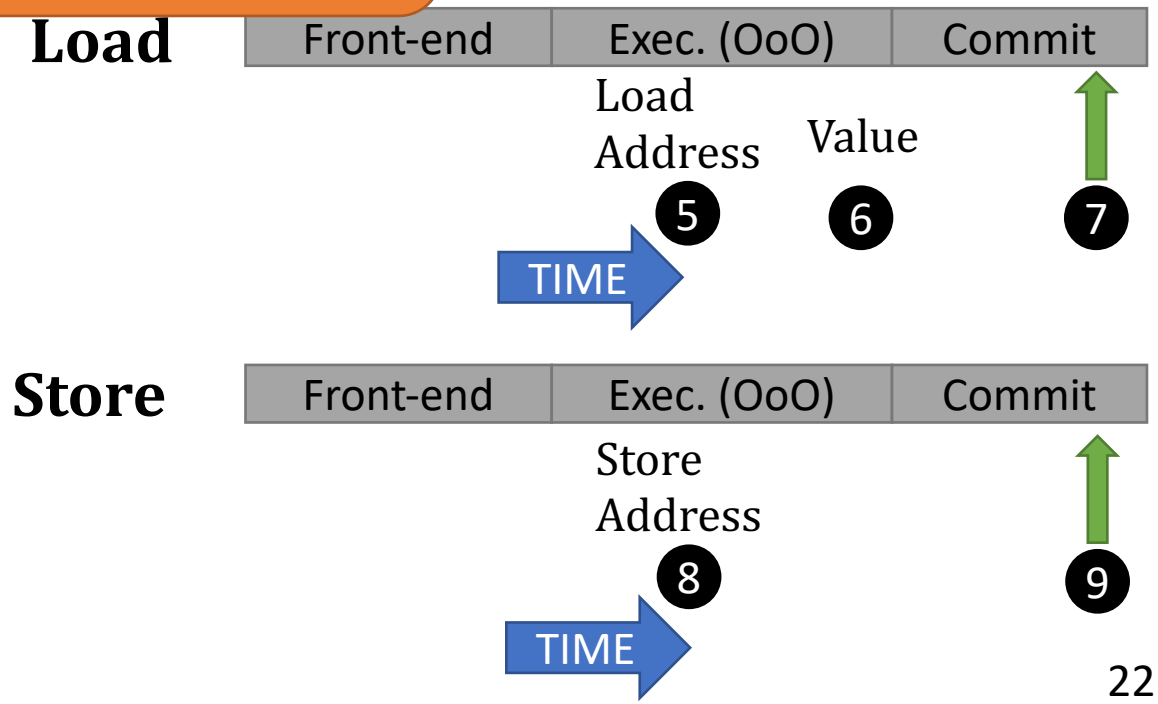




AVPP: Update Overview

- Update the AT at commit of every load instruction ❼
- Update the VT:
  - At the commit of every store instruction, Keeps the VT
  - Should we update the VT at the commit of every load instruction ?





AVPP: Updating the VT at Each Load Instruction

PC	Instruction
40:	LD [4]
40:	LD [8]
40:	LD [12]
40:	LD [16]

Predictor stage: updating

tag	Last addr	Stride
Stride Predictor		

HIT

MISS

addr	value
4	10
8	3
12	9
16	13

Correct predictions : 0

- VT always misses
- No valid predictions

tag	val
4	10
8	3
12	9
16	13

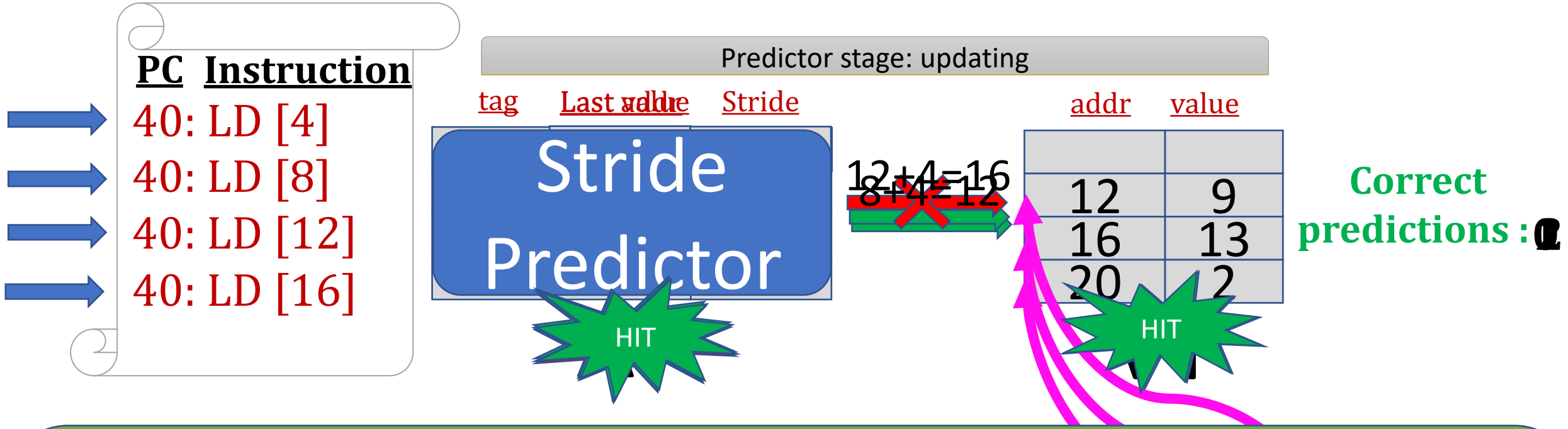
L1 Cache

AVPP: Prefetching Overview

- **Idea:** Prefetch values from the memory hierarchy into VT to increase the VT hit ratio
- The prefetch address is calculated at each address prediction ❸
  - Generated when the prediction is confident
  - Predicted to be accessed by future instances of the same load instruction
  - Dynamic prefetch distance (details in the paper)
- The VT is updated with the prefetched values ❹

The goal of prefetching is to increase the predictor coverage, not to hide memory latency

AVPP: Updating the VT with Prefetching



- Valid Predictions
- VT hits because we prefetch the right values into it

AVPP: Putting All Together

- Leverages the better **predictability of addresses** on load instructions to improve performance

- Two basic hardware structures:
  - **AT:** predicts the address first
  - **VT:** predicts the value next

- **Prefetches** future instances of the load instruction to improve the **VT hit rate**

AVPP Outline

- Motivation and Goal
- Background
- Reducing Hardware Complexity
- Taxonomy of Value Prediction
- Mechanism: AVPP
- **Evaluation and Results**
- Conclusion

Experimental Setup

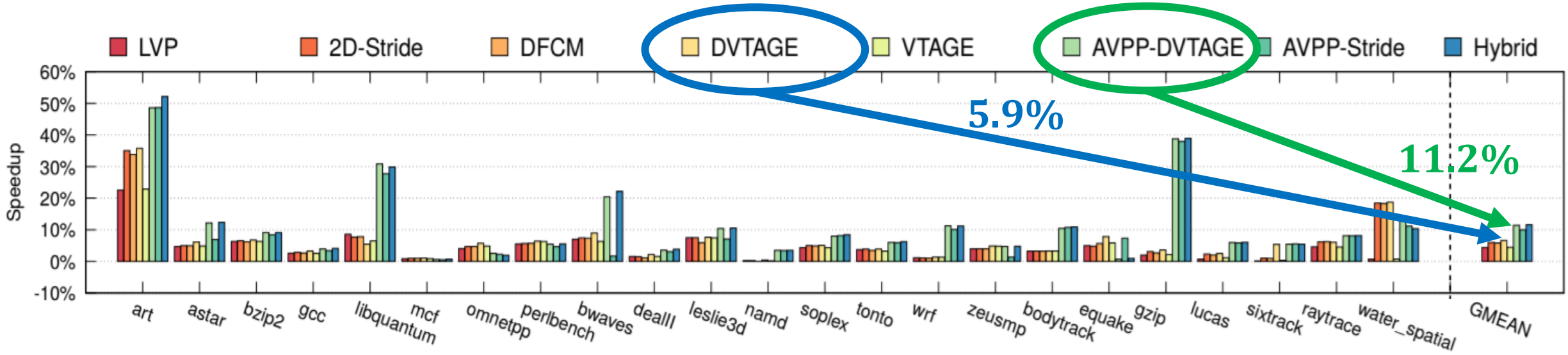
Simulation platform:

- **ZSIM** [Sanchez+ ISCA'13] : 4-issue OoO cores, TAGE branch predictor, 128-entry ROB, 32KB L1 caches, 256KB L2s, Stream prefetchers
- **McPAT** [Li+ MICRO'09] for estimating the energy consumption
- Predictors evaluated:
  - LVP, 2D-Stride, DFCM, VTAGE, DVTAGE, **AVPP-DVTAGE, AVPP-Stride**
  - **512 entries**

23 selected benchmarks from

- **SPEC2000**
- **SPEC2006**
- **PARSEC/SPLASH2**

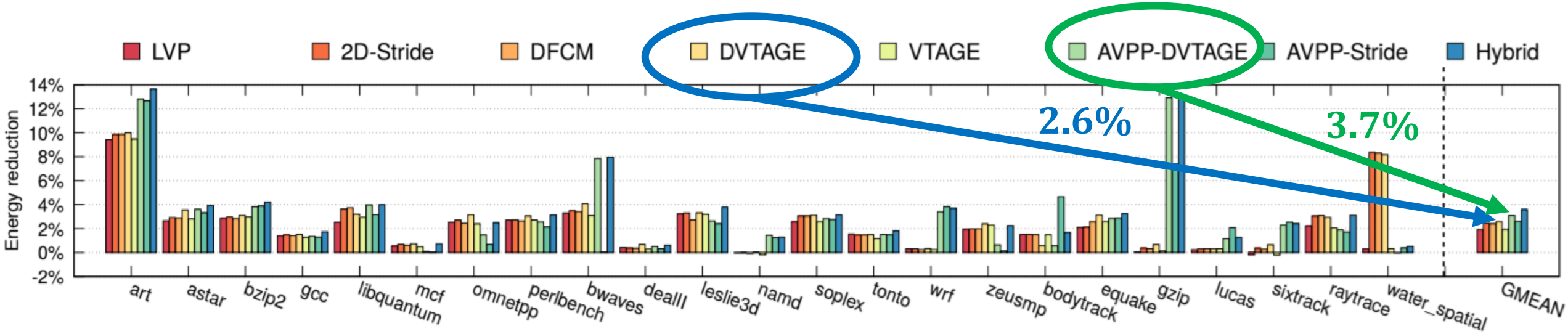
Evaluation: Speedup



AVPP outperforms the state-of-the-art predictors

11.2% speedup over no value prediction

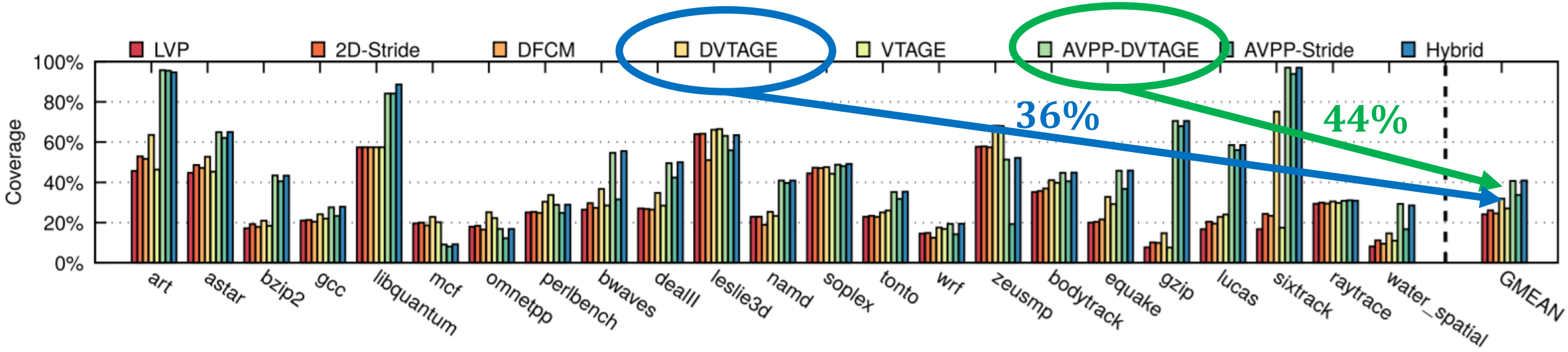
Evaluation: Energy



AVPP outperforms the state-of-the-art predictors

3.7% energy savings over no value prediction

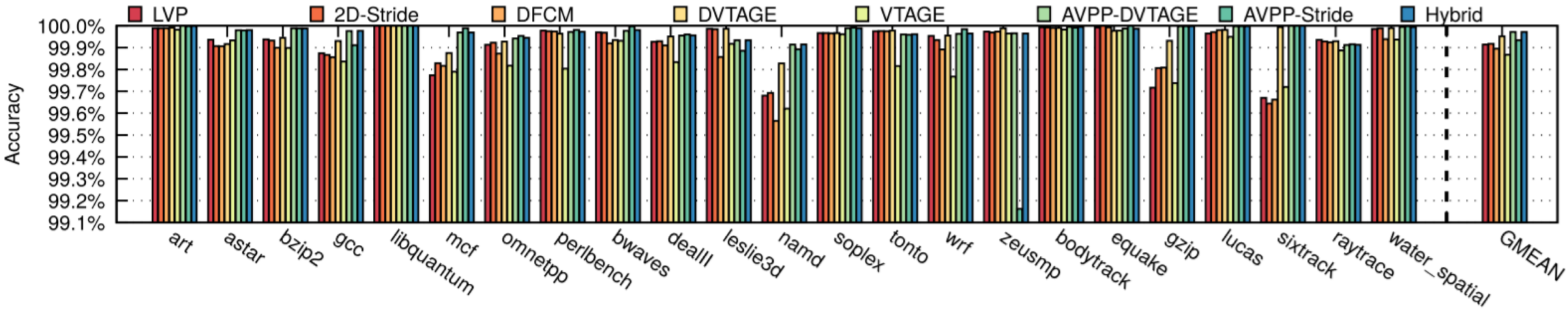
Coverage of the Load Value Predictors



AVPP covers more load instructions than the state-of-the-art

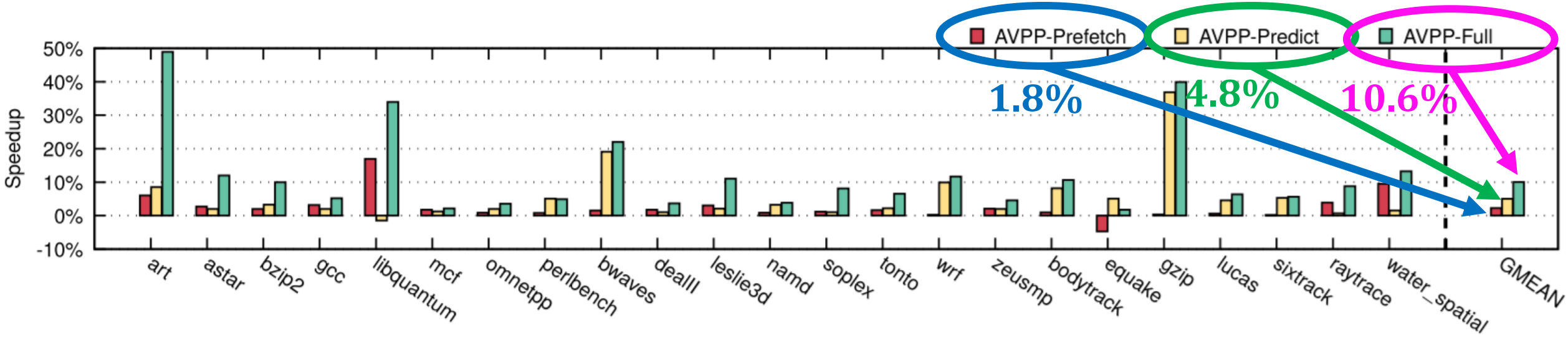
AVPP covers 44% of the load instructions

Evaluation: Accuracy



Accuracy is higher than **99.8%** in all predictors because of the use of the confidence counters

Where are the Benefits Coming from?



AVPP prefetching alone does not provide significant performance improvement

AVPP prediction alone provides only 4.8% speedup

The full AVPP achieves 10.6% speedup

More Results in the Paper

- **Multi-core** simulations
- **Cache hit** distribution
- Distribution of the **prefetch request** that hits in L1, L2 and main memory
- **Sensitivity studies** on
  - Memory hierarchy (larger caches)
  - Load Queue size
- Impact of different load value prediction implementation **policies**

AVPP Outline

- Motivation and Goal
- Background
- Reducing Hardware Complexity
- Taxonomy of Value Prediction
- Mechanism: AVPP
- Evaluation and Results
- Conclusion

Executive Summary

Motivation: single-thread performance is critical for many applications

Problem: improving single-thread performance in modern processors requires techniques that significantly increase the hardware cost

Goal: revisit Load Value Prediction as an efficient way to improve single-thread performance

Contributions:

- We propose optimizations for **reducing the hardware cost of load value prediction**
- We propose a **new taxonomy** of Value Prediction Policies
- **AVPP**:
  - New load value predictor that predicts the **address first** and the **value next**.
  - Increases the coverage of the predictor by **prefetching the value** of a future instance of the load instruction

Results:

- AVPP outperforms all **state-of-the-art** value predictors in the context of load value prediction, and it is **less complex** than predicting all instructions
- AVPP provides **11.2%** system performance improvement and **3.7%** energy savings compared to no value prediction.

AVPP:

Address-first Value-next Predictor with Value Prefetching for Improving the Efficiency of Load Value Prediction

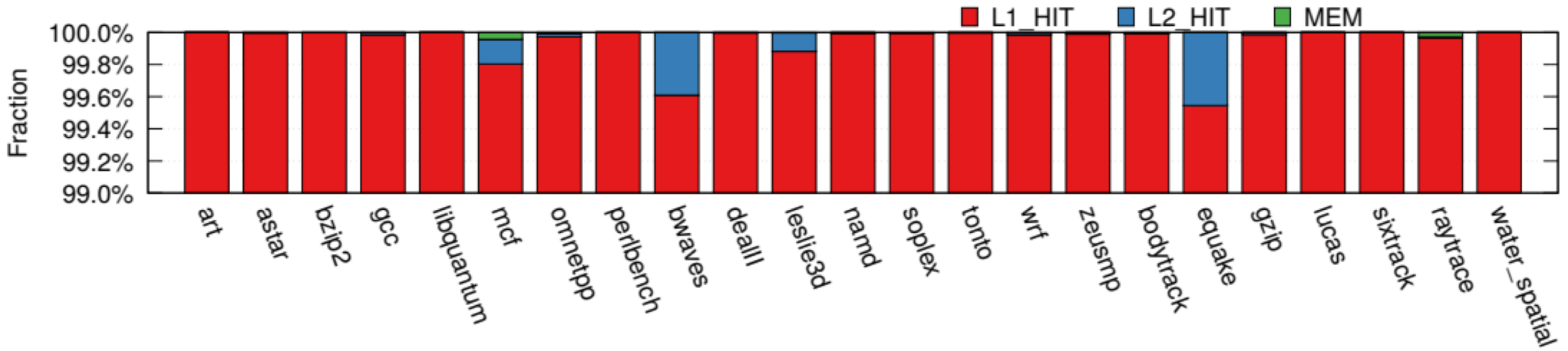
Lois Orosa, Rodolfo Azevedo and Onur Mutlu

SAFARI ETH Zürich

INSTITUTO DE COMPUTAÇÃO
Universidade Estadual de Campinas

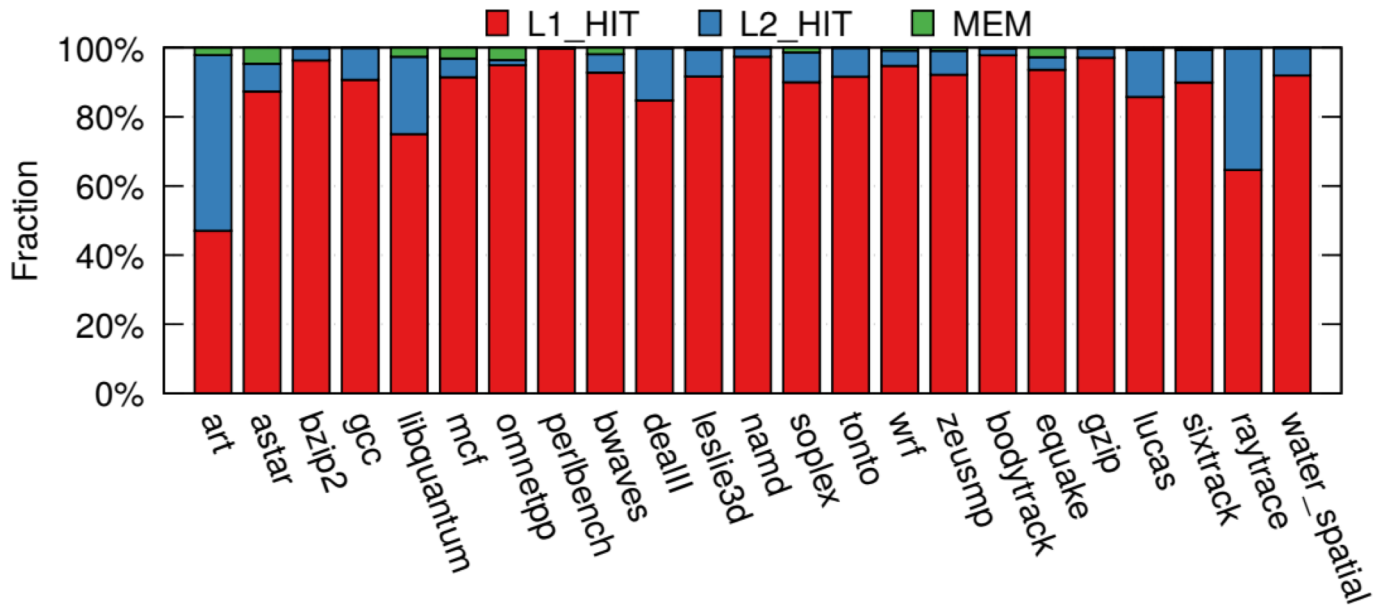
BACKUP

Cache Hit Distribution of the predicted load instructions



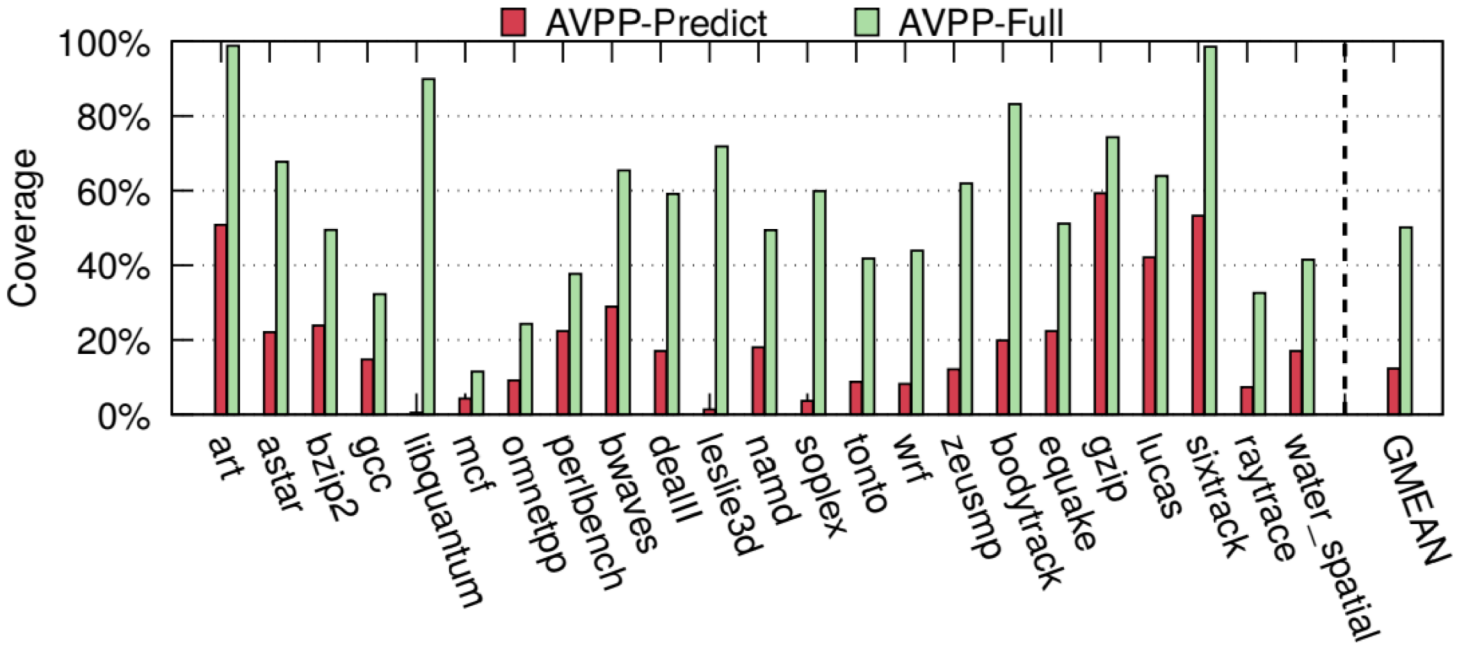
Most of the predicted load instructions hit into L1 cache

Cache Hit Distribution of AVPP-DVTAGE prefetch requests



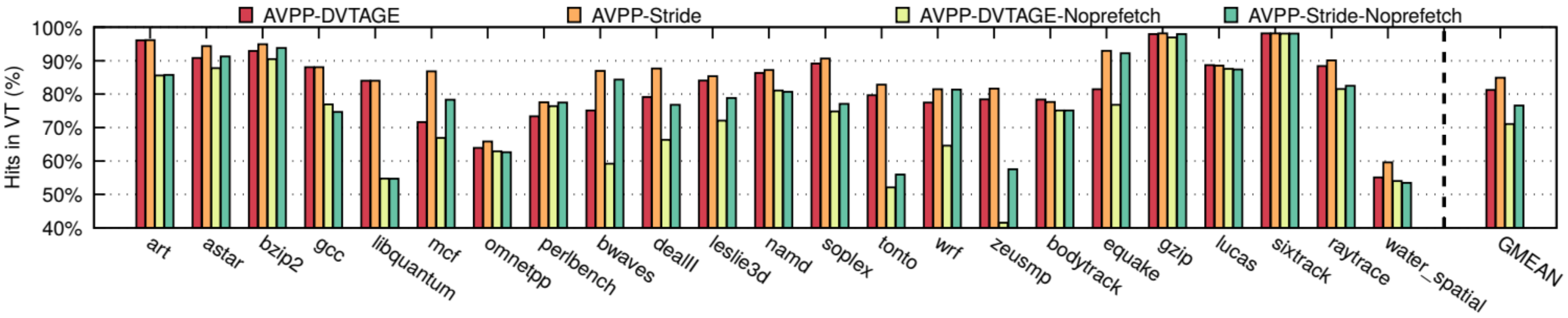
Most of the AVPP prefetch requests hit into L1 cache

Coverage of AVPP-DVTAGE with and without prefetching



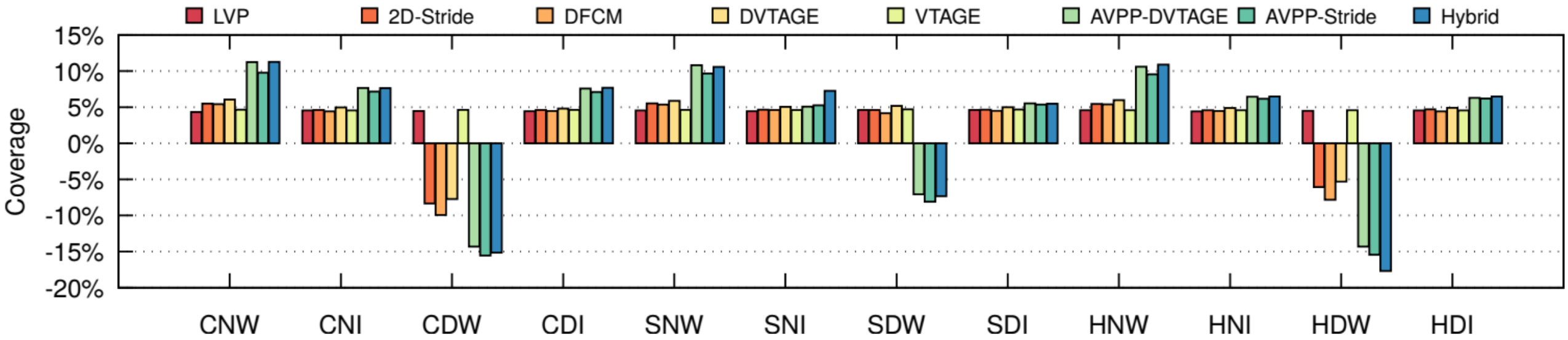
The coverage of the predictor largely improves with prefetching

VT hit rate with and without prefetching



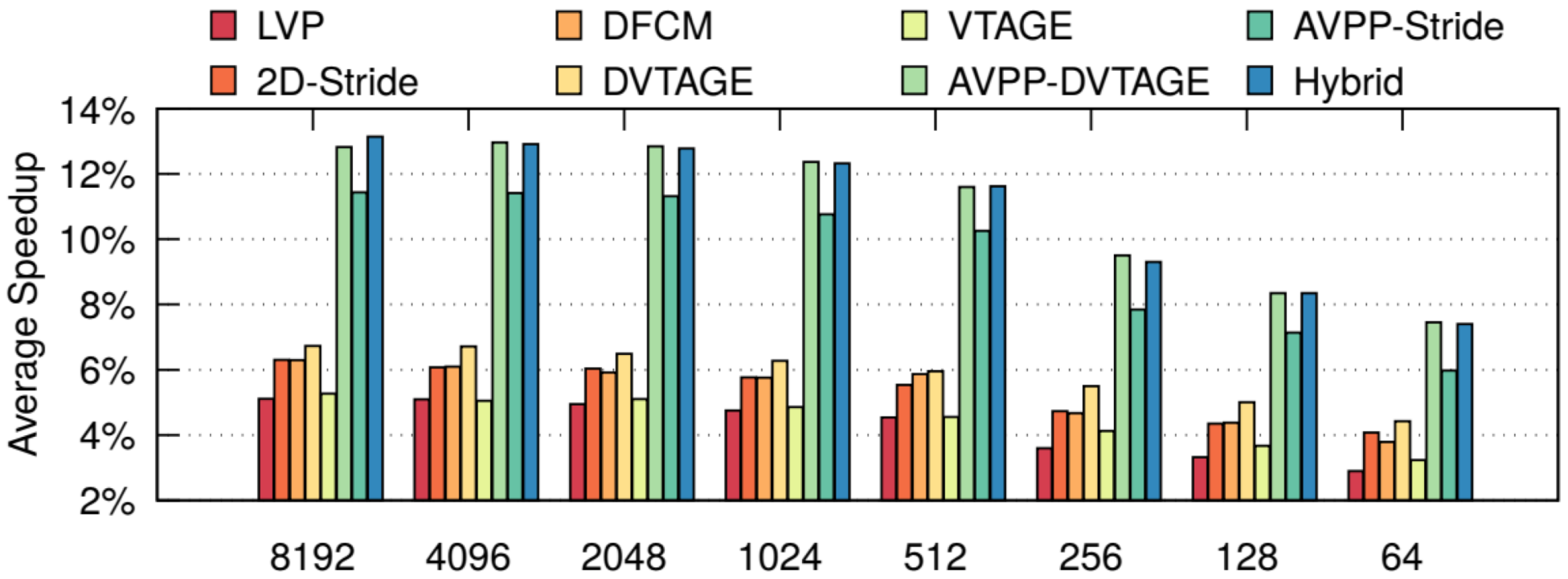
Prefetching increases the VT hit rate

Impact of the Prediction Policies



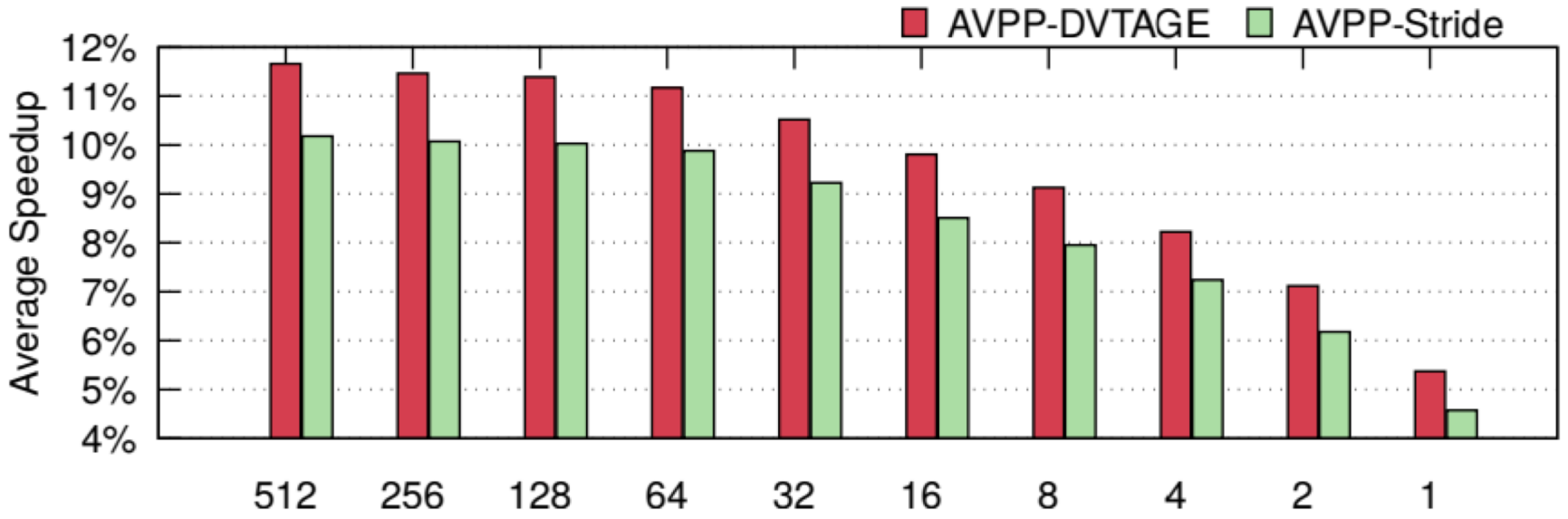
The best policy depends on the microarchitecture

Impact of the Predictor Size



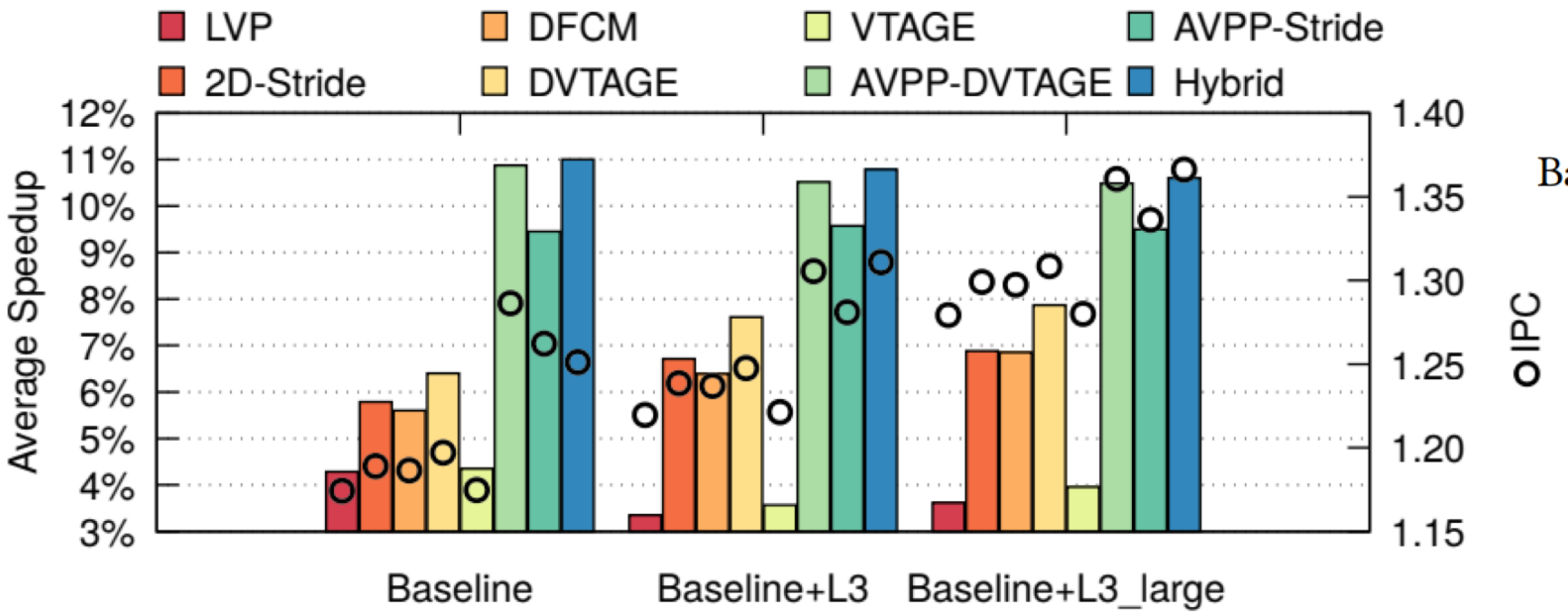
512 entries is a sweet spot

Impact of the VT Table Size



64 VT entries is a sweet spot

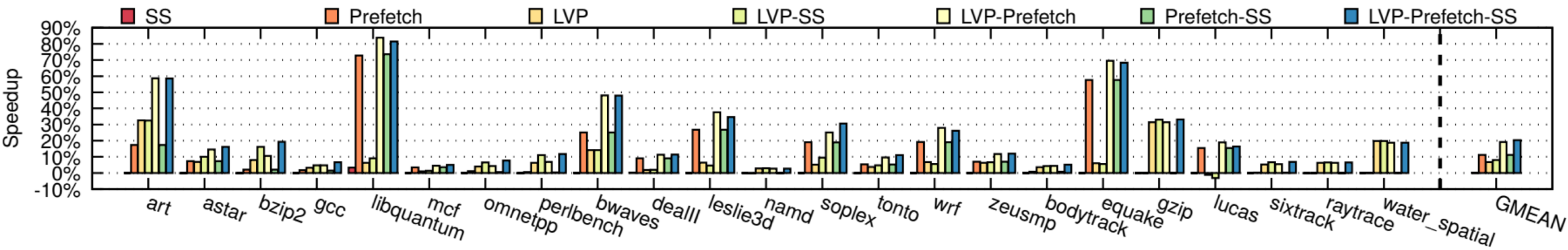
Impact of Cache Configurations



Baseline	L1D&I: 32KB, L2: 256KB
Baseline+L3	L1D&I: 32KB, L2: 256KB, L3: 2MB
Baseline+L3 large	L1D&I: 64KB, L2: 512KB, L3: 4MB

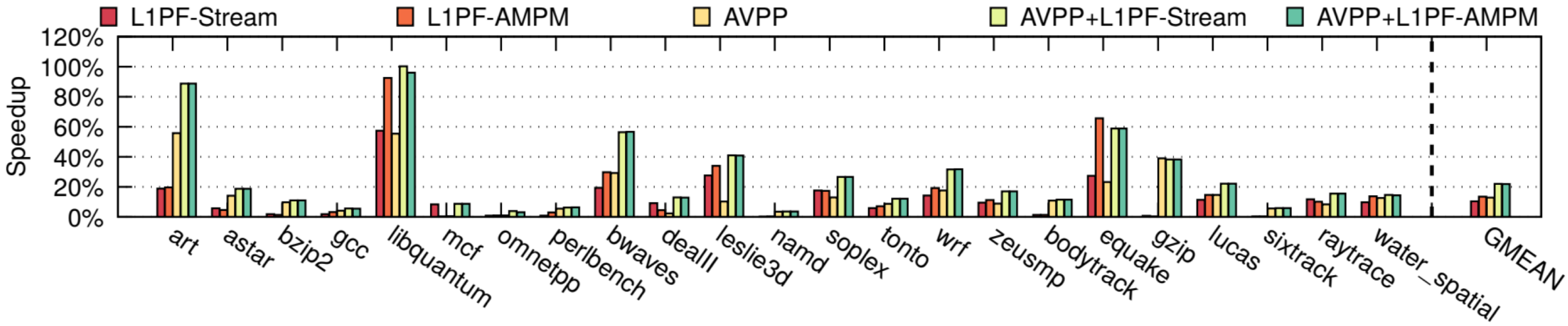
The speedup of load value prediction does not depend much on the cache configuration

Store Set Prediction and L2 Prefetching



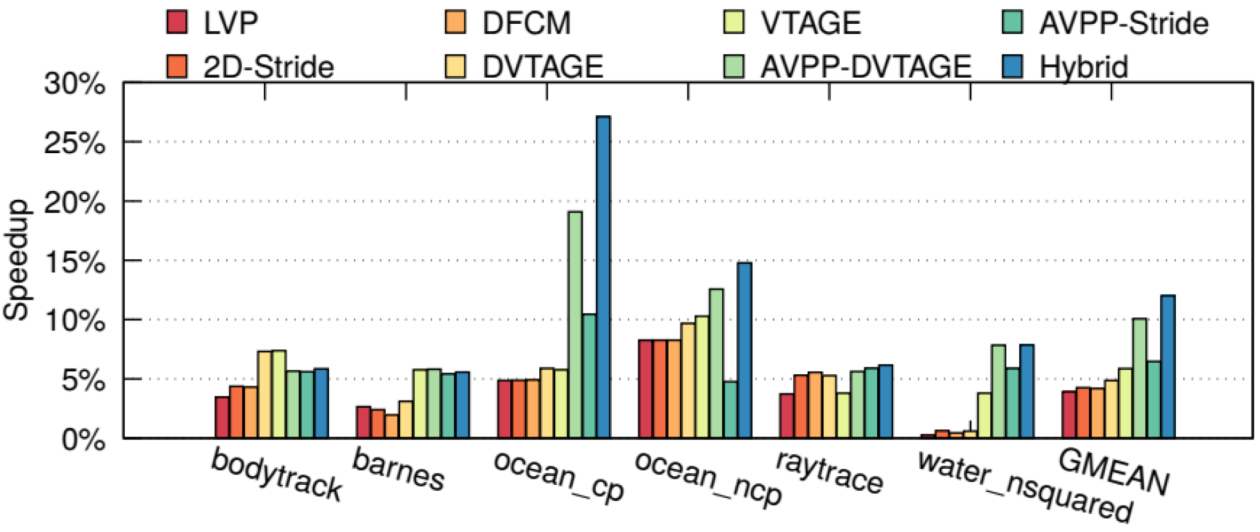
The L2 Prefetcher and Load Value Prediction (LVP) are complementary

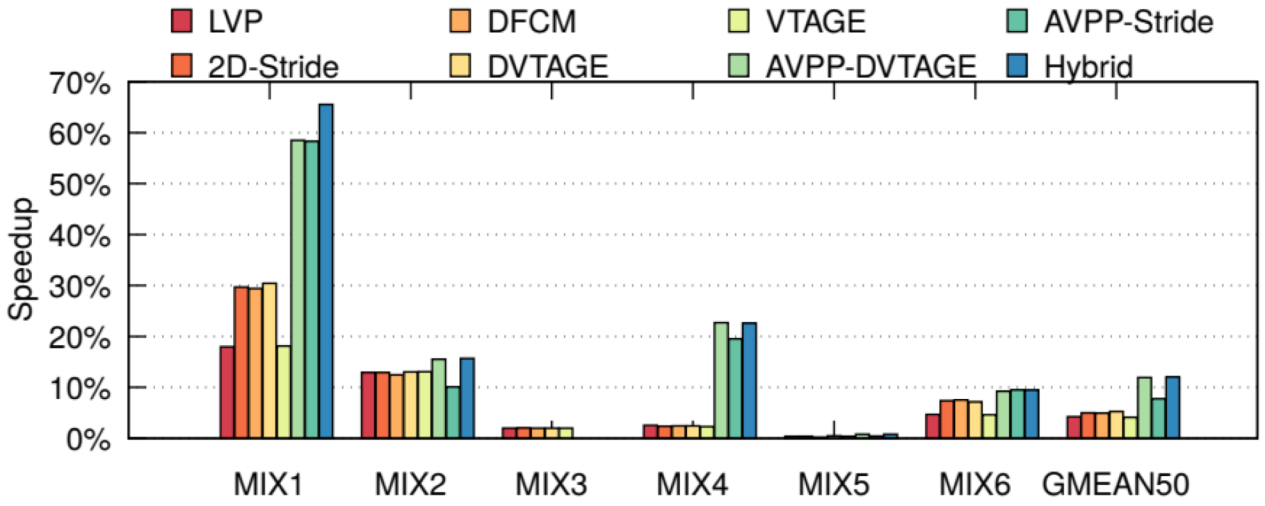
L1 prefetching VS AVPP



When AVPP and L1 prefetcher are used together the performance is additive in many benchmarks

Multithreaded and Multiprogram Mixes





AVPP outperforms previous predictors