Load Value Correlator Predictor (LVCP)

1 Details

The main idea is that TAGE can struggle with hard-to-predict (h2p) branches when the decision depends on data values. TAGE mainly uses branch history (taken / not taken patterns). But for some branches, the same history can lead to different outcomes because the decision is based on a value loaded from memory, not just the path.

LVCP helps by using three key structures:

- **Load Marking Table**: Marks which loads are worth tracking.
- **Load Tracking Queue**: Keeps recent loads (their PC and value) so branches can look them up.
- **Correlation Table**: Uses (branch PC + recent load info) to predict taken/not-taken when there is a strong match.

A key question is: which load should the branch use? LVCP focuses on recent loads. If a load is too old, it becomes less useful and can be moved out of the "recent" set. Another question is: how does it connect a load to a branch? The branch looks back at recent loads and uses a hash of the branch PC, the load PC, and the load value. If that same combination happened before, LVCP can reuse what happened last time (taken or not taken).

2 Load Value Correlator vs Baseline

2.1 Performance Summary

Better performances in how many frameworks of each type:

Framework	Improve/Total	Success Ratio (%)	Total MPKI Improvement
Web	23/26	88.5	-10.4414
Int	20/37	54.1	-14.4377
Fp	5/14	35.7	-2.9306
Infra	9/16	56.3	-2.1943
Compress	1/8	12.5	-0.1299
Media	0/4	0.0	0.0000

Table 1: Performance comparison across different benchmark types

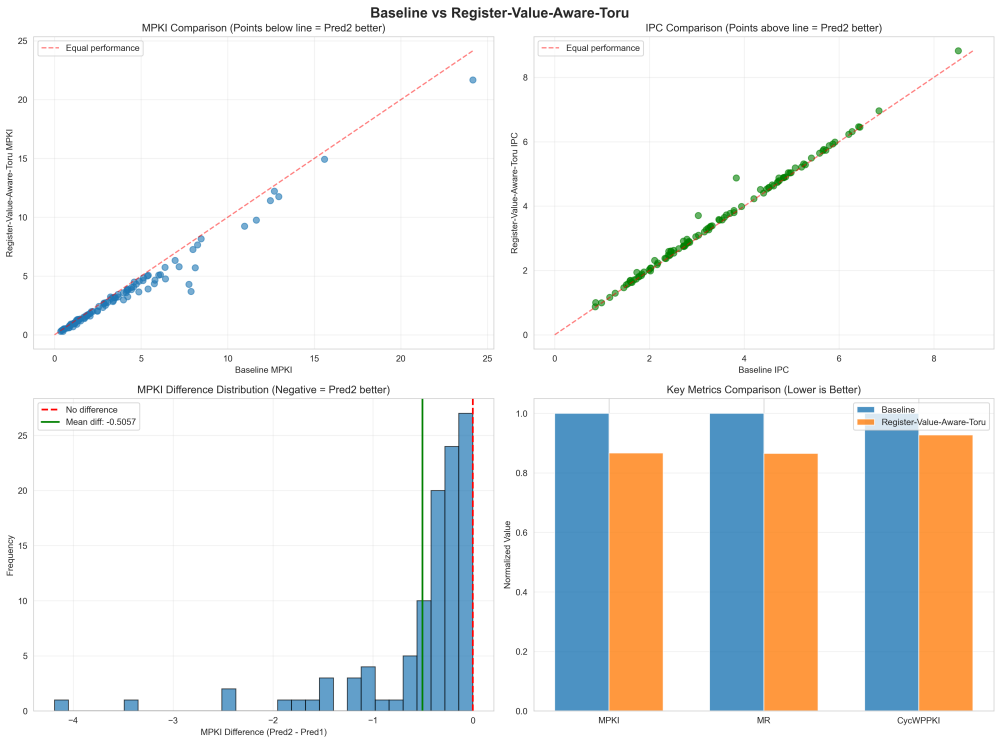

Figure 1: Load Value Correlator Predictor vs Baseline Performance

2.2 Analysis

As usual, it performs better than the baseline TAGE predictor. The biggest gains show up in the web benchmarks. A likely reason is that web workloads often do a lot of checks on input/state that come from memory. That creates many "load then branch" patterns. For example:

```
if (request.type == GET) { ... }
```

This would mean that usually how these JavaScript intensive workloads are structured is that they would need memory access, and this would mean that there would be a lot of loads and stores. The load value correlator is exceptional with loads before branches. In the example above, there would be a load before the branch and this means the load value correlator would shine.

The largest overall MPKI improvement comes from the integer group. That also makes sense: many integer workloads load values from memory and then branch based on those values.

3 Discussion: Why LVCP Helps Some Workloads More Than Others

3.1 When Load Values Actually Help

LVCP works best when a branch decision depends directly on a value that was just loaded from memory. In those cases, the last load is a strong hint for what the branch will do.

If the branch depends on several values, or on values that are computed after the load (math, multiple steps, combined state), then one load value is not enough. In that situation LVCP helps less.

This is why the results are uneven. Web benchmarks improve a lot, while FP, compression, and media often show smaller gains.

3.2 Why Web Benchmarks Improve a Lot

Web code often follows a simple pattern: load a field, compare it, then branch. For example, reading a request type or a flag and immediately checking it.

Because the load and the branch are close together, LVCP can usually find the "right" recent load and use it to predict the branch.

3.3 How This Differs from Register-Value-Aware Predictors

LVCP ties branch behavior to values coming from memory loads.

Register-value-aware predictors look at register values, which can include both loaded values and values produced by computation. That makes them more flexible across different workloads, because they can capture more kinds of data dependence than "just the last load".

3.4 Limits: Timing and Tracking Size

LVCP also depends on timing. If the useful load happened too long before the branch, or many other loads happen in between, the best load might not be the one LVCP picks.

Also, LVCP tracks only a limited set of recent loads. That works for "simple and recent" cases, but not for branches that depend on long-lived or complex program state.

3.5 Summary

In short: LVCP shines when control flow is tightly linked to a recent memory load (common in web-style code). It is less helpful when branches depend on longer computations or more complex state. That is why it is a strong but targeted tool for data-dependent branches.