Register Value Aware Branch Predictor Report

1 Register-Value-Aware-Branch-Predictor

This was a pretty interesting read, and there was a lot of interesting methods that they used.

Runlts uses the TAGE predictor as the foundation, and it consists of multiple tagged tables and a statistical correlator. they have also increased the size of the bimodal predictor to 128k entries.

The key innovation in this predictor is the register value awareness. essentially this predictor uses register values to help predict branches and that is pretty interesting because initially i thought that it would be very expensive to just compare register values everytime. register reads could add to the overhead. but as i read the paper more i was quite intruiged by the techniques that they used. i was pleasently surprised by the quality of branch prediction as it beats andre seznecs 2025 tage sc-l.

Something that i found different/interesting was that they they use the term "digest" when its just a summary. nevertheless, a digest is a 12 bit summary of a 64 bit register value, these digests are used to track correlations between specific data patterns (like loop coutners) and branch outcomes. the digests are generated based on the type of data for example) for int registers the digest includes the count of leading zeros but for fp registers they use the msbs of the expoenent. once the digest are generated they are fed into the statistical register. it then organizes the registers into banks, generates a usefulness table and picks the most useful register. the selected digest is used to look up the predicction in the prediction table. it outputs the actual predicted branch direction.

The register compoenent turned out to provide the largest and the broadest gain out of all the other features. it was the only feature that improved accuracy across every benchmark.

Runlts flags an entry as newly allocated instrad of using u bits to prevent eviction. the system tracks two outcomes for these entries ie succes and waste.

Apart from the register optimizations, runults combines an arithmetic progression with a geometric one, which turned out to be near optima.. they also improved the IMLI. and also adds the call stack compoenent to the statistical correlator. this can differentiate identical branches that might be called from different functions.

They also increased the size of the base predictor as mentioned above.

2 Register Value Aware Predictor vs Baseline Predictor

Okay first things first, in the graph below right i can see that is a significant gain in MPKI. in terms of raw numbers the mpki for the register value aware predictor is 13.28% better than the baseline. there "isnt much" gain in IPC but the Misprediction rate is lesser for the predictor. the IPC improvements tell me that the improvements are central to hard to predict branchges. im pretty sure most of these improvements are because irl most branches are value dependent and this predictor exactly exploits that, since the predictor sits on top of tage with its smart use of register to predict branches it is able to maximize on its gains. from the graphs i can see near zero regressions, it means the the predictor only helps when its useful and when the baseline doenst perform the best. when the values dont correlate, the predictor falls back to baseline behaviour.

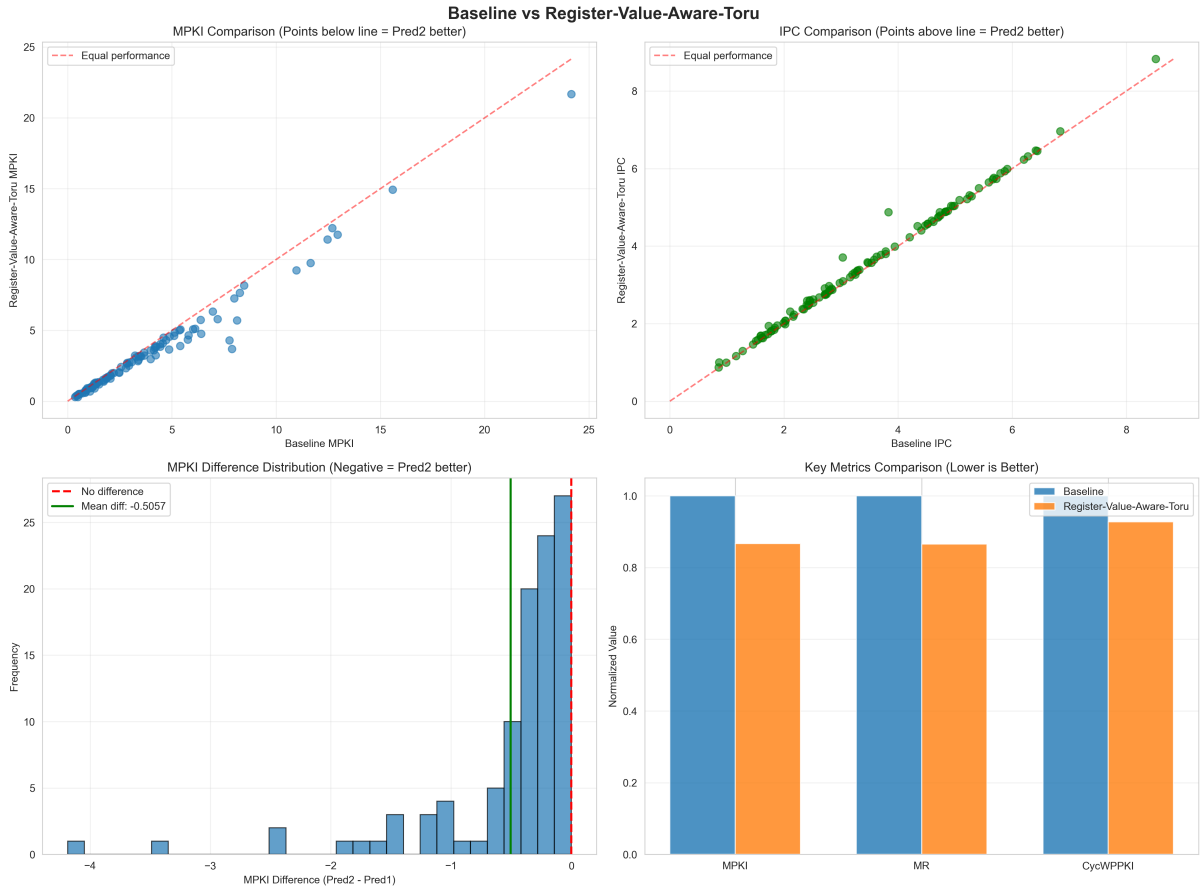

Figure 1: Register Value Aware Predictor vs Baseline Comparison

A good thing is that around 15/112 bencmarks reported "similar" MPKIs so the rest of them have significant or "pretty good" gains in MPKI.

Its not very surprising that a lot of the improvements in the MPKI where the difference is more than 1, belong to the int benchmark. now that intuitevely makes sense since most of the int benchmarks are bemchmarks used to do integer computations, array indexing, control flow decisions etc. but aanother thing to consider is that success rate ie the #imoroved/total is 75% which measn that int benchmarks have a wide vareity so im assuming it does well when it involves branches that use register values, and it turns out that the mpki speedups are very noisy. either having significant improvements or having little to know improvement, mostly leaning towards the imrpovmeemtn side. RVA-Toru performs well on INT because it exploits value.

2.1 Summary Statistics

Ovbiously mispbrpercyc would imrpvoe if mpki improves. moreoverall, benchmarks show any sort of visible improvement (even thoguh we might call them similair due to inherent error margins and noise).

another thing that was quite noticable that MPKI reductions occur on traces with already high baseline MPKI, suggesting that the predictor sort of "clutches" on branches where history based correlation is weak. this kind of indicates that the register value aware predictor is able to exploit value correlation where history based predictors fail.

The variance in INT benchmark improvements reflects the heterogeneity of integer workloads. While value-dependent control benefits significantly from register correlation, pointer-driven and indirect branches limit the effectiveness of value-aware mechanisms, leading to uneven

Metric	Baseline	Register-Value-Aware-Toru	Difference
MPKI	3.8080	3.3022	↓ 13.28% better
MR	2.9679	2.5681	↓ 13.47% better
IPC	3.4636	3.5450	↑ 2.35% better
Cycles	14824660.9143	14401963.4667	↓ 2.85% better
BrPerCyc	0.4203	0.4309	↑ 2.54% better
MispBrPerCyc	0.0100	0.0089	↓ 10.23% worse
CycWPPKI	161.6047	149.9141	↓ 7.23% better

Table 1: Summary statistics comparison

but predominantly positive gains.

one another good thing about the predictor is that it is able ot defer to the baseline tage predictor when register value correlation is weak or absent, preventing performance degradation. this is evident from the lack of significant regressions across benchmarks.