Darin Soeung

Computer Arch. SimpleScalar Report

lou957

I'd like to preface I will be including all of my data through a google docs link at the end of the report in order to give a better understanding of my thought process as well as provide the data I am presenting in another medium.

After obtaining SimpleScalar I began step 2 of part 3 Project Procedure, which required me to evaluate the performance impacts of the different branch predictors we were given. I began to execute each branch predictor across all 5 benchmarks given in order of, CC1, Perl, Go, Comp, and Anagram. Parsing through the results after each command I grabbed the Simulated Instructions per Cycle (IPC), Simulated Number of Instructions, Simulated Number of Branches, and the number of misses received from the specified branch predictor. The simulated number of Instructions as well as the number of branches were of course constant to make comparisons easier between different branch predictors. The simulated IPC as well as the number of misses varied between the different branch predictors. The Perfect branch predictor consistently and obviously had the highest IPC per Benchmark, while Combining, 2Lev, and bimod were consistently within .1 cycles of each other. As seen in Figure 1 a bar graph similar to the example one given to us, you can see the consistencies between the results and the IPC's. Perfect is almost always above and beyond the others except for in Anagram and Comp, where as in comp the result is miniscule compared to the others so the result is hard to see but Perfect is at least .045 IPC's above the rest. On the opposite end it's apparent to see the inefficiencies of the taken/nottaken predictors. On average the taken/nottaken branch predictors had the lowest, by a far margin, IPCs.

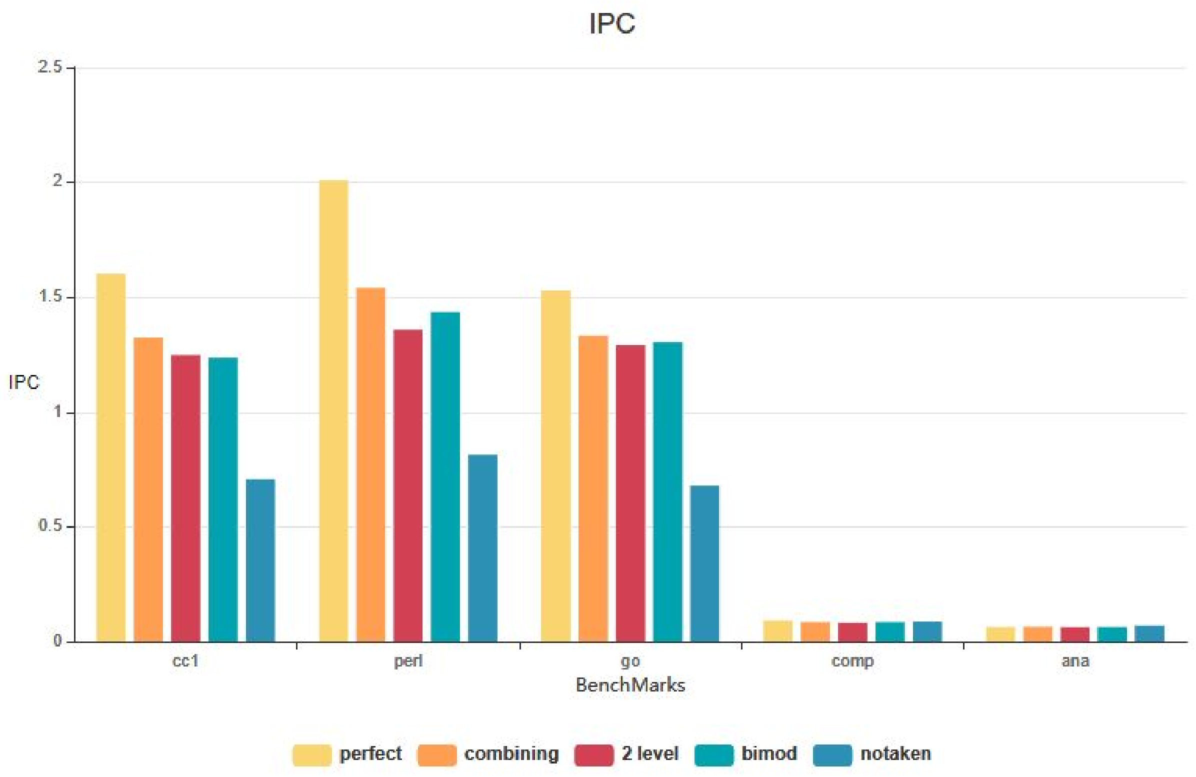


Figure 1

In the next Figure, Figure 2, I display the resulting branch misprediction rates of each branch predictor alongside the specified Benchmarks. The first thing one may notice is that the branch predictor Perfect is not found alongside the other predictors. But as there are no mispredictions by the Perfect branch predictor I chose to not include it alongside the others as it's miss prediction rate would be 0. You may also notice the Miss Prediction Rate (MPR) is rated in %, which I obtained by computing the number of misses by the branch predictor, divided by the simulated number of branches, then multiplied it by 100. Unlike the Perfect branch predictor the MPR of the taken/nottaken branch predictors was massively high. Averaging across all benchmarks at a 67.73% MPR, which in comparison to the other branch predictors as shown in Figure 2, a substantially larger rate than the average. This concludes my data on step 2 but I'd also like to preface that I used all predictors on their default settings.

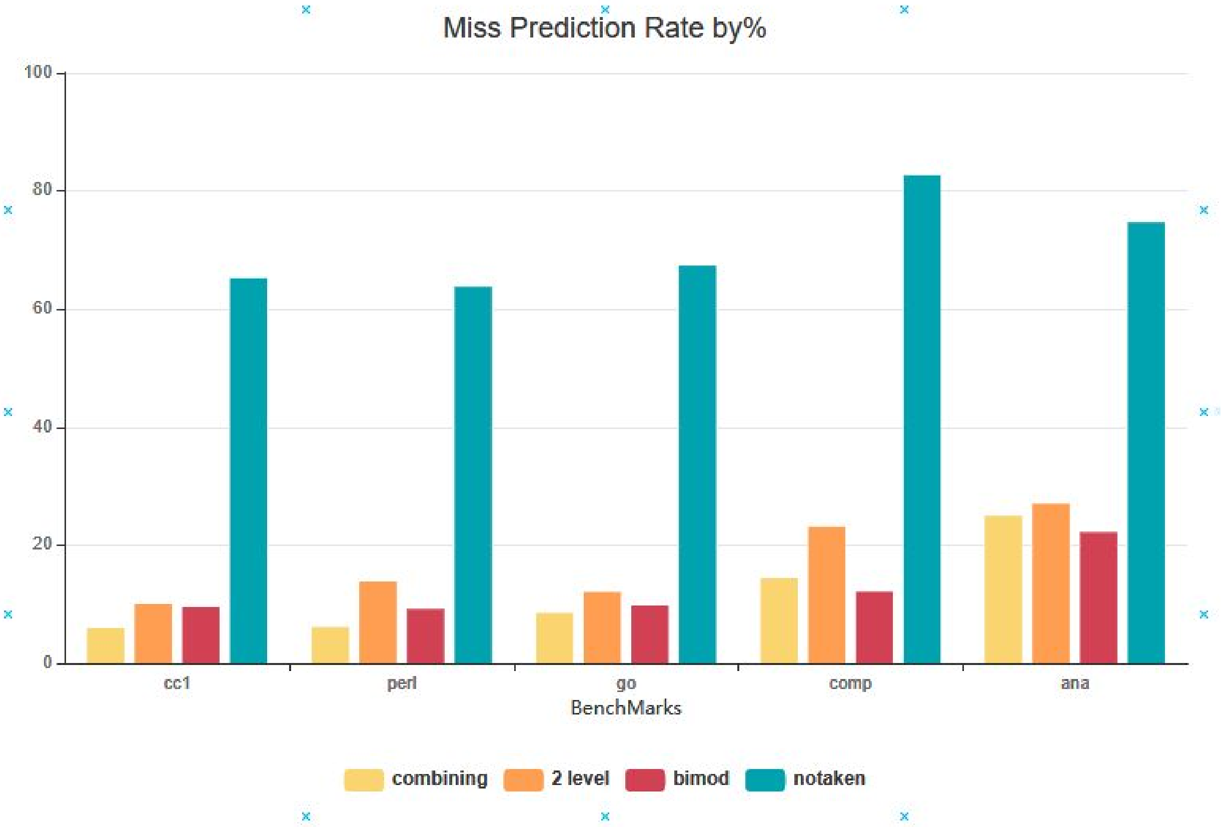


Figure 2

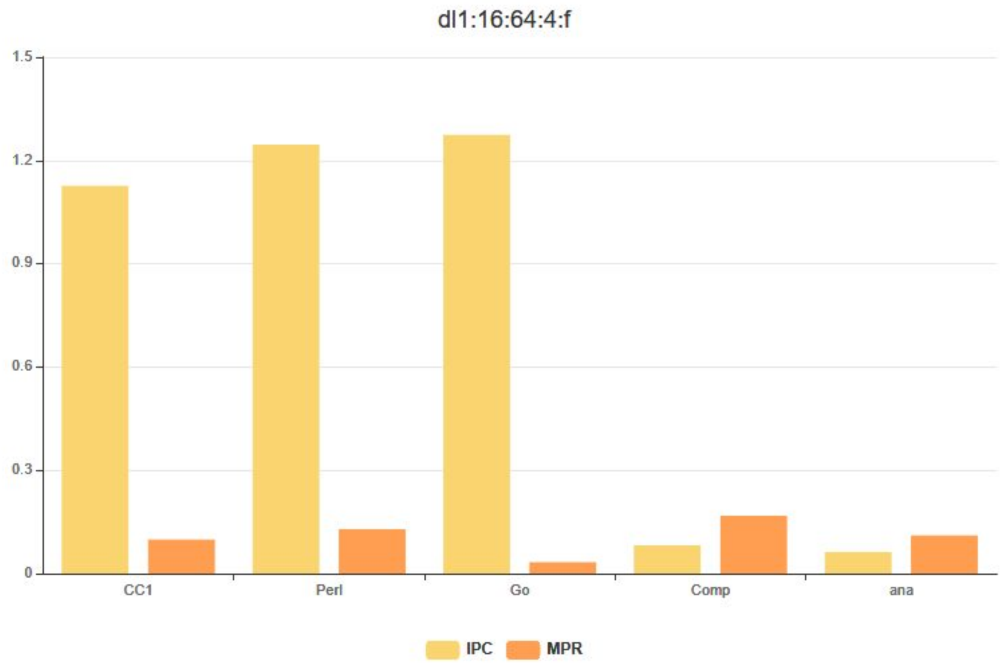
Now on to step 3, evaluating the impact of L1 data cache configurations on performance. Taken into the initially given note I worked my way down progressively from larger to smaller cache sizes. Below I have a figure for each of the 5 cache configurations I tested out with the specific configuration as the title. Please note I have given the Cache Miss rate at the given rate and not in a percentage in order to keep it on the same graph and avoid a clutter of 10 graphs straight just to show simple data. Also note I have messed up on all my graphs and written MPR instead of CMR for Cache Miss Rate, this is prior to me having made them and I am sincerely sorry please overlook that and see MPR as the Cache Miss Rate.

The first cache configuration (Figure 3) I used was similar to the one given through the project instructions. I simply used it to get a grip on what I was doing along with the format given to change the cache configuration. I started a bit big as I wanted to use it as a basis for when I could see the bigger changes as I got to a smaller cache. Here the cache miss rate was relatively low, if you calculate the CMR (MPR) as 100*given CMR, you can see that all of them were below 20% with relatively high IPCs over 1 per cycle for CC1. Perl. and Go. For the bigger cache sizes as shown in Figures 3 and 4 the CMR barely breaks 30% and IPC's remain relatively high near at least 1 Instruction per cycle.

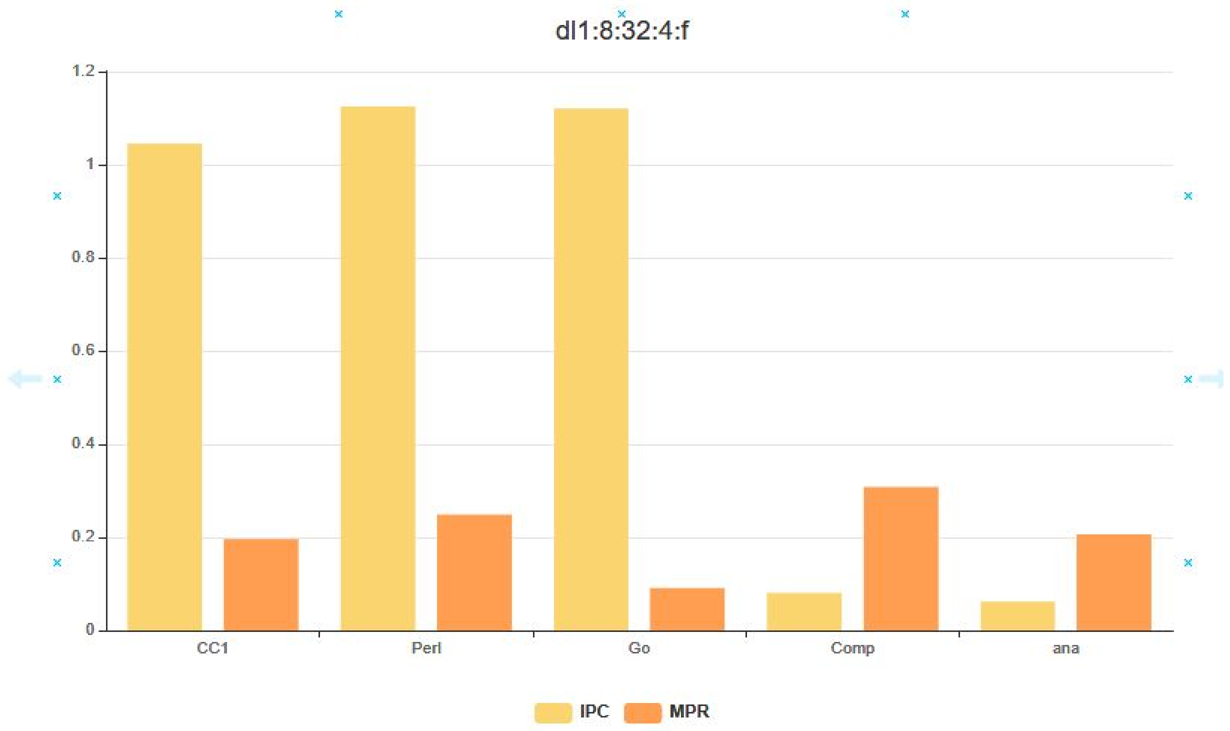
However for the smaller cache sizes Figures 5-7 you can see a great increase in CMR and decrease in IPC. The CMR rises to around 40% for all benchmarks while IPCs slow down to below .9. Here we can already start to see how smaller cache sizes greatly affect our performance and it only gets worse the lower we go.

Pushing the boundary on figure 7 you can see I really try to minimize the size of the cache to a block size of 8 and only 2 sets in the cache and a 4 way set associativity. I use a random block replacement strategy as the number of blocks to keep track of is really low. This resulted in a range from 58% to 80% Cache Miss Rate and an IPC average of .22 instructions per cycle, with the highest being .53. You can see how punishing a smaller cache size is on impacting performance on this scale.

Comp Arch Simple Scalar Report



Figures 3&4



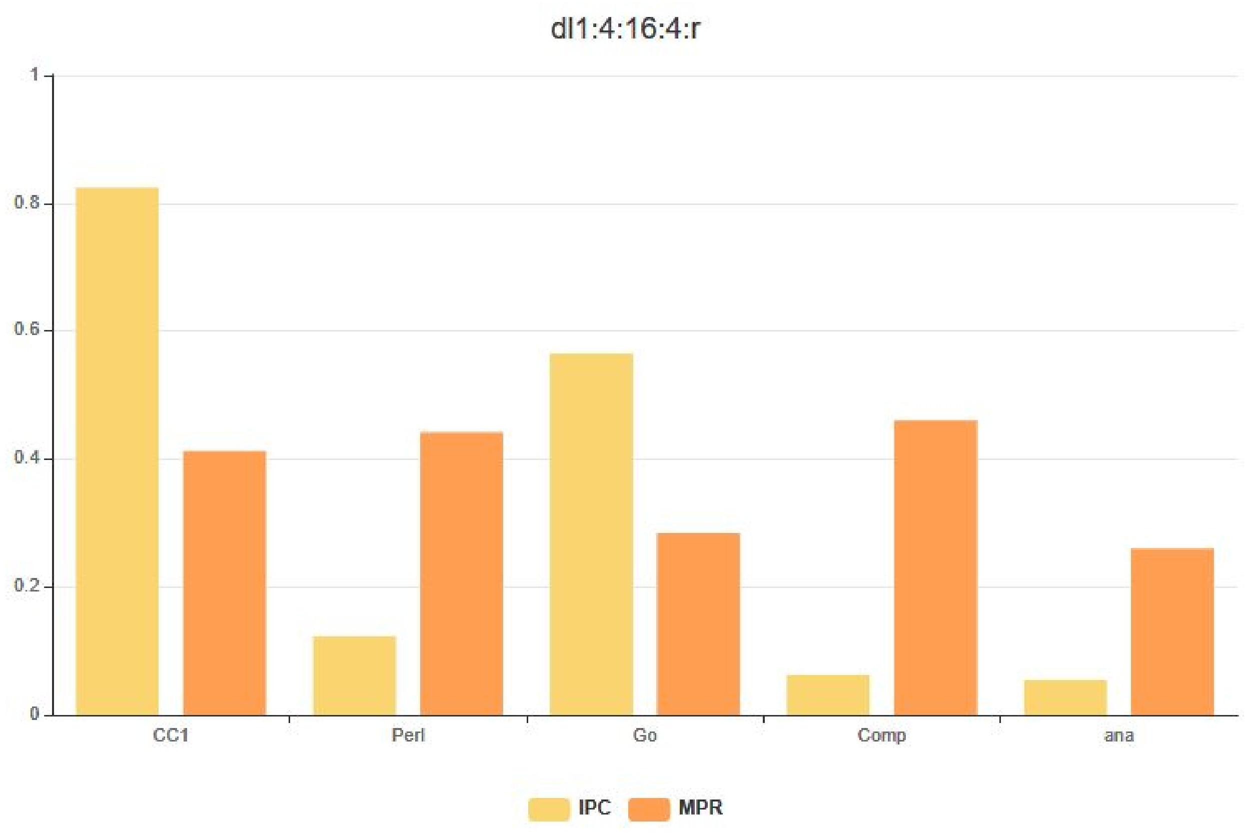


Figure 5

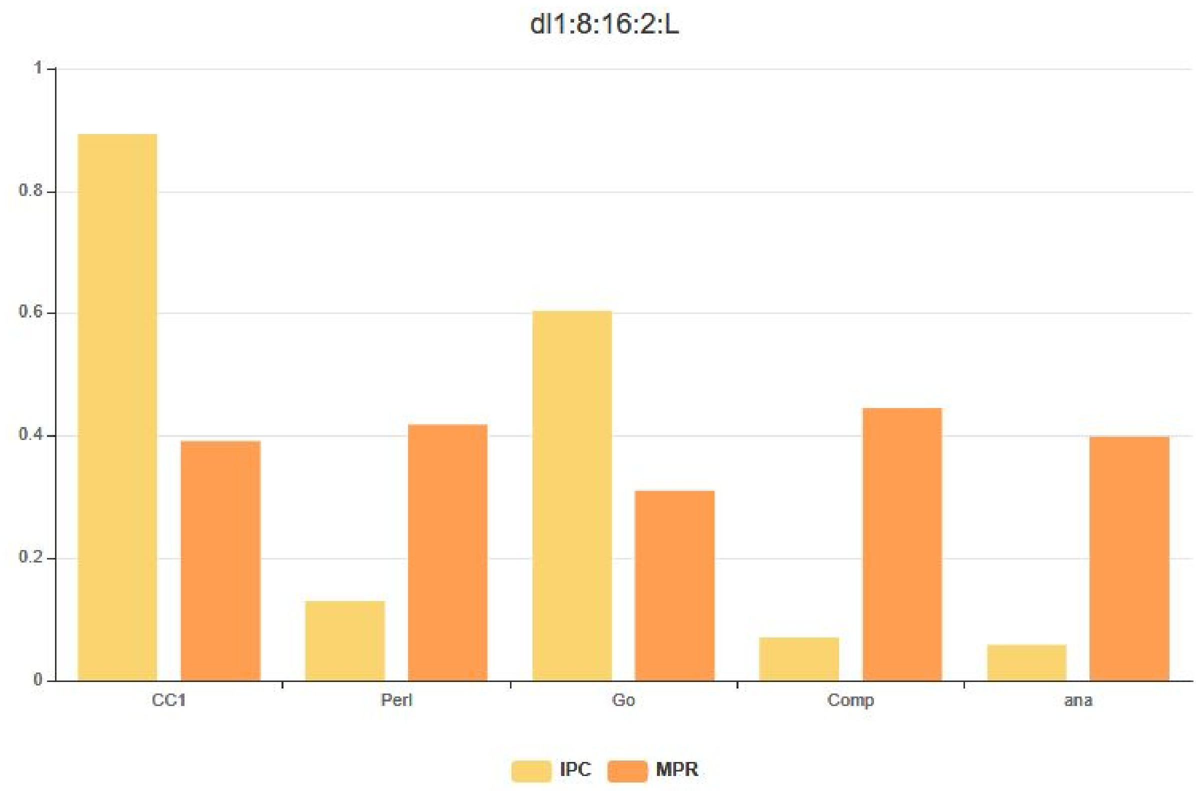


Figure 6

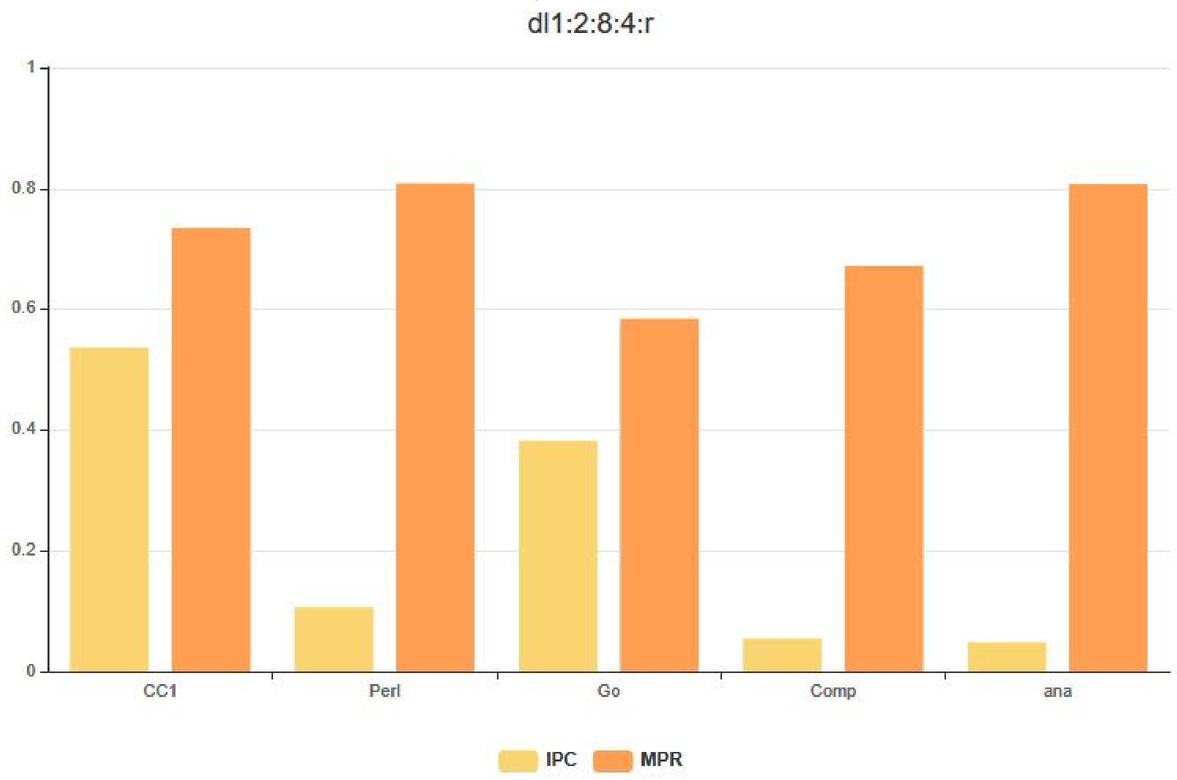


figure 7

On to step 4, the task of freedom and exploration, I followed the instructions under 4a and went to check configuration of Branch Target Buffer BTB. Which means you guessed it, more graphs! I ran BTB alongside all benchmarks and sent it into a data table shown in Figure 8. In Figure 8 I have compared the IPC of the BTB configuration 16 sets and 4 associativity alongside the original 2lev IPC.

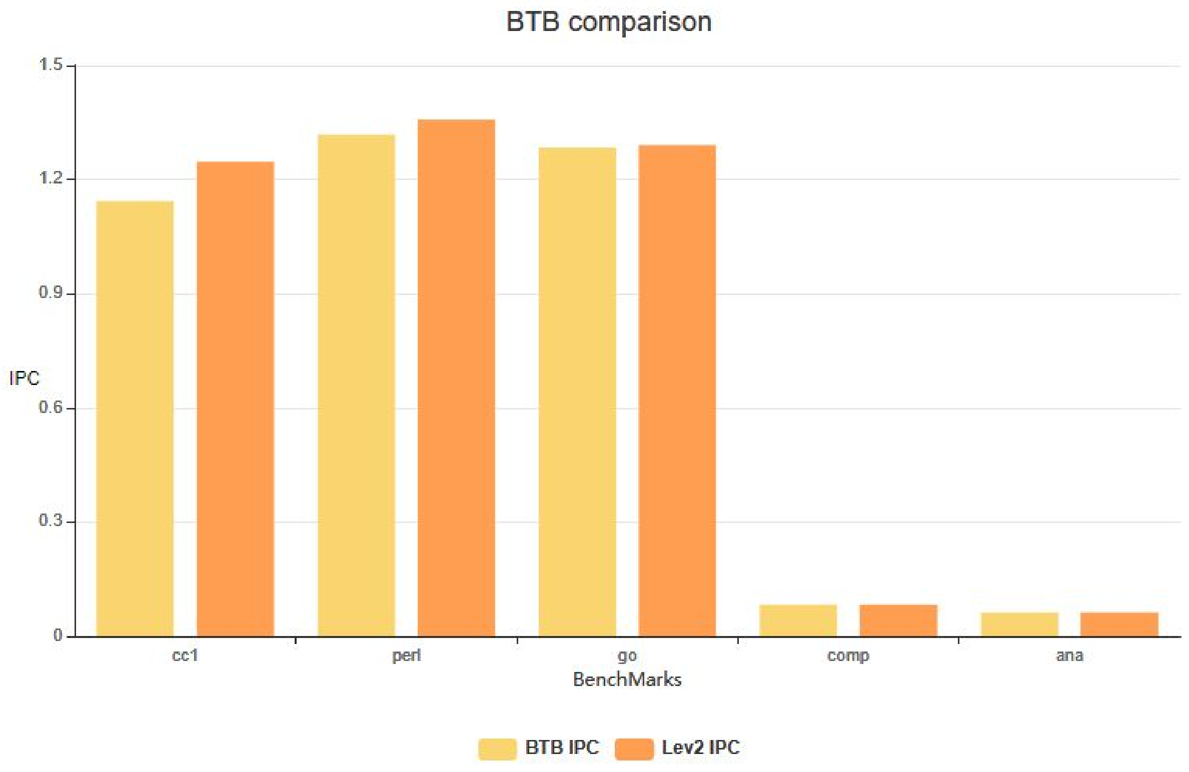


Figure 8

You can see that the results with the smaller configured Branch Target Buffer had IPC's that were on average lower than their Lev2 counterparts. So you could probably guess that with a smaller Branch Target Buffer we have less existing branches in the buffer in which we can choose our branch target. With a smaller buffer we get a limited amount of choices on these targets and they can be cycled out based on the given predictions. So with less room to work here you can see it does show a visible effect on our performance albeit a small one.

Comp Arch Simple Scalar Report

Here is the data I used in this project as well as the website I used to make my graphs.

https://online.visual-paradigm.com/

https://docs.google.com/spreadsheets/d/1oMxNlRqcftF2BgHvvgZVPUw1I_BbqBRoWW-IOT9Q
Y3Q/edit?usp=sharing