The implementation of our two-level Branch Predictor is an imitation of the gshare

In order to gather global information, a global history table collects the branching decision of all recent branches regardless of their respected program counter. The table is essentially a 1D left shift register tuple, shifting the latest branching decision from the LSB. In our design. global history tale is a 5-bit left shift register.

Local information is collected by local history table. Branching decision recorded in the local history table are separated by program counter, so the size of the table would affect the granularity of the branching record sampled. In order to reduce the table size, my implementation limits the local history table to size 32 x 5. The significant 5 bits are use as index to the table. Note that gshare engages global and local information by exclusive or the two value, so their length must match.

A two-level branch predictor incorporates a branch likelihood table to make the decision of branching or not at the fetch stage. The size of the branch likelihood table is once again a tradeoff between size and granularity. We set the size to 16 to strike a balance between table size and effectiveness.

The entry size of the branch likelihood table is :2^ global history table \* saturating counter size.

the formula shows the global history table size, which equals the entry size of the local history table is exponentially related to the BLT size. It infers that the concept of infinitely enlarging the local history table to store more branching history is theoretically impossible, because an even larger branch likelihood table must be stored. The sweet spot of the size of record to hold is somehow a nice topic to further dive in.

We are interested in testing how three different attributes of a BPU design would affect the Coremark performance of Aquila. One-level predicter against two-level predictor, different BPU record size, and the size of the used saturated counter. The accuracy of each BPU unit should incorporate in the comparison to provide a better insight of how distinct models work. Sadly, there is some technical issues so only Coremark score and total cycles ran are shown here. Despite the fact that the overall accuracy of our prediction unit is not shown, we may say that the BPU design is better if and only if it has a better Coremark score in our experiment since all other factors are controlled and stayed constant.

One-level branch prediction collects only recent history from the execution of a local area, whereas two-level branch prediction has a larger vision. Two level prediction would

Table size of BPU remembers the PC address that was confirmed as a branch instruction. The larger the size of the BPU, it could record more address thus larger portion of the branch executed could take advantage of the branch predicting mechanism. The entry size of the BPU table size is linearly related to the overall memory size of the BPU table. In our implementation, a register array is used to record the branching address and the block ram is used to store the branching address.

We are interested with the relation with BPU performance and the BPU branch table size. Experiments with configuration of 32 entries and 64 entries are conducted. From

The saturating counter records how likely the branch will be taken and the default size of the saturating counter is 2-bits. The original implementation of the counter in Aquila is by finite state machine. However, by implementing a 3-bit saturating counter, FSM may not be a good idea. A real counter is directly written in the RTL level, since it’s more convenient for the designer. The initialization state of a 3-bit saturating counter is a little different than a 2-bit saturating counter. If the branch is first observed taken, state 2’b11 is initialized. However, in the 3-bit saturated counter, state 2’b110 is initialized. 3-bit counter need more incorrect predictions to change the branch prediction, so if the initialization value is 2’b111, we may suffer from more incorrect branches. As if the prediction is correct the next time, we could reach 2’b111, indicating the branch is almost always taken. How the initialization could affect the performance of the branch predictor is another topic to further dive in.

Would switching to 3-bit saturated counter lead to better performance? Tabl shows the statistics collected from the experiment. One-level branch predictor and two-level branch predictor with 64 BPU table entries are tested. We may observe that adopting a 3-bit saturating counter compromises the performance of the one-level branch predictor, dropping the Coremark score from 84.837 to 84.78. As for two-level branch predictor, adopting 3-bit saturated counter boosted the Coremark score from 74.874 to 77.296.

One-level branch prediction collects only recent history from the execution of a local area, whereas two-level branch prediction has a larger vision. Two level prediction requires more resources to implement, structures like local history table and pattern history tables are large 2D register arrays. So theoretically, two-level branch predictor should perform better than one-level branch predictor.

From table 3, our experiment seems to show a different result. Two-level branch predictor seems to perform worse than one-level branch predictor. The best performing one-level BPU is with 64 entries of branch table size and 2-bit saturated counter, scoring 84.837 in Coremark. Two-level BPU with 64 entries of branch table size and with 3-bit saturated counter scored only 77.296. The data collected from our experiment is incoherent with the theory.

Two-level branch predictor is much more complicated than one-level branch predictor, so fine tuning becomes more important. Grid search shall be taken to find the best configuration like global branch history size, local history table and pattern history table entries. I believe after some fine-tuning, two-level branch predictor models would eventually outperform one-level branch predictor.

Despite the existence of a branch prediction unit. Most branch instructions at the fetch state are still not recognized by the BPU. These branches are automatically assuming branch not taken, and the mechanism works quite well. The default one-level 2-bit saturated counter BPU isn’t doing a good job when it comes to forward branching, guessing only 55% correct. By switching to two-level branch predictor, the performance in our experiment actually dropped, different from what we expected. 3-bit saturated counter fits the two-level branch predictor model, raises the Coremark score by about 1.5. Maybe further fine tuning shall be done to reveal the true capability of a two-level branch predictor.

We verified improvements in IPC by adding branch prediction unit to Aquila SoC. We found in Aquila’s default BPU, about 90% of all branches are not found by the BPU and the default not taken decision is taken. Among those recognized branches, BPU only guess 55% correct when it comes to forward branches. Our implementation of two-level branch predictor performing worse than one-level BPU, only score77.297 in Coremark. Where one-level BPU scores 84.837. 3-bit saturated counter fits the two-level branch predictor better than 2-bit counters, improving Coremark score about 1.5.