Progress Report 3

Qingyi Wang (qwang83)

Mingyuan Han (mh24)

Shitao Liu (sl53)

For this checkpoint, as usual, we split the main task into three different parts, and each member was responsible for implementing and testing his/her designed as planned in our last roadmap. Specifically, Shitao implemented the whole cache system and tested the overall performance of the whole design; Qingyi designed and implemented the local branch history table and global branch history table, and based on them developed a tournament branch predictor. Afterwards, Qingyi also developed the m-extension for our CPU using the simple add-shift algorithm. Mingyuan finished both basic and advanced hardware prefetching part.

Cache System

Our L-2 Cache system supports customisation. Specifically, with the offset fixed to 5, the user can choose between 4-way and 8-way design in the top-level file as well as the number of index to extend the storage of the L2 cache system. In detail, we implemented the multi-way cache using the pseudo-LRU algorithm, and, to speed up, we didn’t use a complicated always\_comb logic to find the LRU; rather, we integrated the hit/miss of each way into an array and passed it to the LRU tree, and the LRU tree updated its LRU code and returned a masked array indicating the LRU. Since the number of possible LRU code is finite, we assigned each situation a masked array to return.

Besides, our L-1 cache system also supports customisation to extend its storage. Since the total available design resource on our FPGA board is limited, we found that an asymmetric L-1 cache system could most efficiently boost the performance. With a relatively larger I-Cache and a smaller D-Cache, we can boost the performance by around 20%.

Eviction Write Buffer

The EWB speeds up the write-back process. The buffer can switch the “read” and “write” order in the write-back stage by storing the dirty data and its address inside the buffer and sending a read request to the memory first. And the buffer writes the dirty data to the dirty address after the reading is finished, successfully hiding the latency by changing the RW order.

Branch Predictor

We designed a tournament branch predictor for our CPU to predict the next PC. The tournament branch predictor consists of a local branch predictor and a global 2-level branch predictor. The local branch predictor contains a table of 2-bit predictors indexed by a combination of the PC values and the history of conditional branches at those PC value. The global branch predictor includes several pattern history tables and a branch history register to choose among these PHTs. It records the outcomes of the last several branches and combines with the PC to perform a history table index. An FSM is then used in the tournament branch predictor to choose between the local branch prediction result and the global branch prediction results. We have also parameterized our branch predictor. Both the number of the sets and the number of the last branches can be parametrized. According to the performance counter, by using 1024 sets and 2 last branches outcomes, we can achieve 88% prediction correctness in average, while we only got 25% prediction correctness in average using static-not-taken branches. We wrote our own testbenches to test the branch predictor and also test all the given testcodes.

M-Extension Support

The M-extension consists of a divider and a multiplier. The divider uses a classic long division scheme, and the multiplier uses a simple add-shift scheme. We successfully ran the comp2\_m.s and the performance is 2 times faster than the comp2\_i.s. We wrote our own testbenches to test the multiplier and divider and also tested all the given testcodes.

We were planning to use Wallace Tree to implement the multiplier. However, when we tried to use the Booth Encoder to save the area, we encountered some bugs and didn’t have enough time to solve them. We’ll try to re-implement the multiplier later if time allows.

Prefetching

Basic prefetching is mainly used for instruction cache. It is a one block look ahead prefetcher that always prefetch next cache line, when there is a miss. It could always improve the hit rate of instruction cache, but the performance may not always be improved. Also, we found that one block look ahead prefetcher works terrible for data cache. We also tried to implement stride prefetcher to improve the performance of data cache, but there is still some bugs remaining. We are planning to try our best to finish the stride prefetcher by the final deadline.

Timing

We kept an eye on the timing test along the way implementing our design. Our design can run at 103MHz.

Diagram, schematic

Description automatically generated

Road Map 3

Qingyi Wang (qwang83)

Mingyuan Han (mh24)

Shitao Liu (sl53)

Our design can successfully pass all three competition test cases right now, and it can also pass the design baseline. However, there is a tradeoff between power consumption and running time. We are planning to test all combinations to find the optimal one for the design competition. By the end of this week, we shall finish the preparation for the project presentation to be held next week.

After the project presentation, we are going to write our final report and finish it by May 10th.