ADL/Fast System Implementation of Alpha 21264 Tournament Predictor

Written & Implemented by Brendan Beecham for CS4431

The following is a report regarding the implementation of the Alpha 21264 tournament predictor using the ADL/Fast system.

To create the ADL/Fast system implementation, I utilized the provided documentation regarding the Alpha 21264 tournament predictor and the course lecture slides. The tournament predictor consists of four total integer arrays that are used to determine the best branch-prediction method for the current branch. The resource allocation specifics of the integer arrays could be easily determined by reading through the 21264_articulo paper. In this resource, the Advance Branch Prediction section detailed the dynamic branch prediction scheme where the size of the Local history Table was $1024 \, x \, 10 \; = \; 10240$, the Local Predict was $1024 \, x \, 3 \; = 3072$, the Global Predict was $4096 \, x \, 2 \; = \; 8192$, and Choice Predict was $4096 \, x \, 2 \; = 8196$. Additionally, the global predictor and local predictor use 2-bit and 3-bit counters, respectively, so an additional step multiplying the above values by a byte-size and dividing by the counter bit-size. The final constant definition worth-noting are the global and local branch history sizes which are 12 and 10 respectively.

The foundation of the branch prediction logic lies in the procedure *predict_taken_not_taken,* which was present in the provided gshare predictor and was used as a starting point for the tournament predictor. The tournament predictor has a few notable differences from the gshare predictor while retaining the structure and purpose. The conditional check for btb_hit remains in place since the predictor will not proceed if this isn't the case. Notably, the differences are caused by the addition of using the value choice predictor at the current branch index to determine whether local or global predictor will be used for the branch.

The implementation of the choice predictor is quite trivial. The choice_pht integer array is accessed indexing with the global_history_bits after a btb_hit is registered. This global_history_bits calculation is the same as the pht_index in the gshare predictor. The value that will be accessed in choice_pht will be a 2-bit saturated counter value. If the value is 0 or 1 then the local predictor will be used in this branch. If the value is 2 or 3 then the global predictor will be used in this branch. Regardless of which predictor is used, the predictions for both methods will be calculated and stored in global 32-bit integers to assist in updating values in the EX-stage of the pipeline. Once the EX-stage is reached and only if branch_input is present, choice will be updated based on three conditions:

1. If the stored global prediction result is equal to the branch_input.
2. If the stored local prediction result is equal to the branch_input.
3. If the current branch-index of choice_pht is within the bounds of its 2-bit saturating counter.

Depending on the state of the above conditions, the current branch-index of choice_pht will either be decremented, incremented, or unchanged. This process allows the branch prediction to adjust to branches that are being repeatedly mispredicted by either local or global in favor of the other prediction method.

The final major change between gshare and the Alpha 21264 tournament predictor is the implementation of the local predictor to be used when global predictions are being mispredicted repeatedly. This implementation uses both the local branch history table to maintain the last 10 branch histories and index these histories into the local predictor for the prediction results. Similar to the method used to index both choice_pht and the global predictor in gshare. The local history table is indexed using a logical and (&) of the most significant bits of the current-branch's PC with a branch_history_table_mask, equivalent to one less than the size of the branch history table. Next, the value accessed in the local branch history table is masked with the local branch history mask equal to one less than the size of local_pht. At this point, the last result of this branch has been stored in the control register local_pht_index and the prediction can be made. As with gshare, a case switch statement using the btb[btb_index].b1_branch_type will determine the type of branch. If the branch is a conditional_direct_link, then the local prediction will be set to the first bit of local_pht_index.

The implementation of the global predictor is exactly the same as that of gshare with changes only made to the branch_global size (as stated earlier in the report). After both predictors have made their prediction and choice has determined the method for this branch, the history values will be updated accordingly. The global history is identical to gshare while the local branch history simply substitutes the global branch_gr value for the current index of the local branch history table.

In order to update the values in local_pht and global predictor, an if-else statement in whether branch_input was received. This is done only if the branch is a conditional_direct_link. If this statement is true, the values of the current index in both the global and local are separately compared to their respective maximum saturated counter values. For local, this maximum value is seven since it is a 3-bit counter. For global, this maximum value is three since it's a 2-bit counter. The value at the current index will be incremented if the current value is less than the respective maximum. In the case when branch_input was not received, the current index value of both predictors is compared to zero. If the values are greater than zero, they will be decremented. This process updates the predictions according to their performance.

It is worth noting that a global 32-bit control data register *measuredPrediction* is used to record the branch prediction method that choice uses. This register is set to zero at the start of predict_taken_not_taken, then updated to either 1 or 2 if local predict or global predict is used, respectively. The value of *measuredPrediction* is used to calculate the total accuracy for both local and global predictions in the EX-stage.

The results of the tournament predictor can be analyzed by juxtaposition to the results of gshare by running the entire Spec-95 suite on small (other sizes for the specs would be more accurate, however small was chosen for the lower run-time). As a comprehensive statement, the tournament_hit_rate (the accuracy of the tournament predictor) is varied only slightly in percentage to gshare_hit_rate. The percentages with both gshare and tournament can be seen

below. These results were compiled from the benchmark file produced by running the simulation with the ADL file as input.

Spec-95 benchmark	gshare percentage	Alpha 21264 percentage
129.compress	96.89%	97.24%
126.gcc	86.16%	86.94%
130.li	95.72%	92.97%
132.ijpeg	89.83%	90.94%
134.perl	96.97%	96.67%
147.vortex	97.76%	97.65%
099.go	78.70%	76.04%
124.m88ksim	96.89%	96.39%

In the table above, it is apparent that the benchmarks that have a measurable percentage improvement from the Alpha 21264 tournament predictor are 129.compress, 126.gcc, and 132.ijpeg. Next, we'll look at the CPI results for each.

Spec-95 benchmark	gshare CPI	Alpha 21264 CPI
129.compress	1.093551	1.092646
126.gcc	1.329137	1.326160
130.li	1.278699	1.291657
132.ijpeg	1.211833	1.207533
134.perl	1.282915	1.283918
147.vortex	1.147109	1.147526
099.go	1.236488	1.246281
124.m88ksim	1.250242	1.253008

As was seen with the percentages, the CPI has varied improvements depending on the benchmark being tested. The Alpha 21264 tournament predictor sees improvements with the 130.li, 134.perl, 147.vortex, 099.go, and 124.m88ksim benchmarks. It is worth mentioning that the benchmarks that saw improvement in CPI did not see hit rate improvements and vice versa.

Overall, every Spec-95 suite benchmark sees an improvement in either hit rate percentage or CPI utilizing the Alpha 21264 tournament predictor. It can be assumed that the larger the test benchmark that was run (i.e. running test or train instead of small for the simulator), the greater the difference between the two branch predictors. This emphasizes the importance of choosing the right branch prediction method for the right instruction set architecture to fully take advantage of the hardware in use. While my implementation of the Alpha 21264 tournament predictor was not perfect, it shows the benefits of dynamic branch prediction and demonstrates the fundamental functionality using the ADL/Fast system.