ADL / FAST Alpha 21264 Branch Predictor Project

Matthew Spencer

12/11/2020

CONTENTS

INTRODUCTION

This project comprises building a simulated Alpha 21264 [1] using the FAST / ADL computer architecture simulation language. This is done by manipulating the simulated 6-stage-pipeline CPU Instruction Fetch (IF) and Execution (EX) stage that has a pre-installed Branch Target Buffer (BTB). The Alpha implementation is done by programming a tournament predictor in the IF stage that dynamically chooses between two primary predictors to predict whether a branch is being taken or not taken during the execution of a MIPS program, a local history predictor and a global history predictor. The EX stage then reports back whether the prediction was correct or not by updating the corresponding variables to better predict the next branch by toggling between the two predictors.

MATERIALS

- Linux Ubuntu 20.04.1 LTS [2]
- FAST / ADL Computer Architecture Simulation Language [3]

THEORY

Using the FAST /ADL language, an Alpha 21264 predictor can be implemented by having a 4K-sized choice tournament 2-bit predictor indexed by a 12-bit global history branch buffer choose between a local and global predictor. The 1K local 3-bit saturating predictor would be indexed by a 1K local history table, which in-turn is indexed by the current program counter address provided by the installed BTB buffer, and the 4K global 2-bit predictor is indexed by the 12-bit global history branch buffer alongside the choice predictor.

The Alpha 21264 implementation with its dynamic choice predictor should be able to predict branches on average more accurately than the global predictor and local predictor acting isolated on their own while executing multiple data sets which contain branches that can benefit from utilizing both. This implementation should also lower the average overall Cycles-Per-Instruction (CPI) of the selected data sets as well due to the combining of branch predictors. [4]

PROCEDURE

To build the Alpha 21264 architecture, a local and global predictor must be built into the IF stage of the 6-stage-pipeline ADL script. Figure 1 displays the local history predictor, and its local history table is used to store the 10 most recent branch histories indexed by the local BTB address supplied by the program counter. The size of the local index history table is 1K, which allows for up to 1024 addresses to be indexed. The total bits needed from the local address is then $log_2(1024) = 10$bits. This is taken from the program counter's address bits starting from the least-significant-bit, (LSB) going from bit 0 to bit 9. This is represented in the ADL/FAST language as `my_pc.[0:9].` The prediction of either taken or not taken for the branch is determined by the most-significant bit of the 3-bit saturating counter.

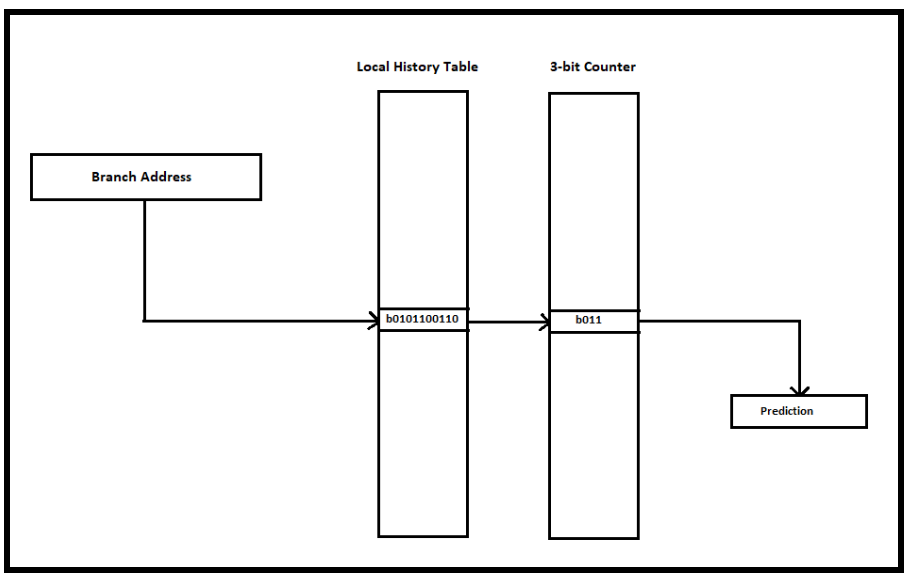

Figure 1: Local History Predictor

The address index represents a 10-bit integer that saves the history of the last 10 branches regarding that particular address, whether it was taken or not taken. This history is saved by left-shifting a 0 bit (not-taken) or 1 bit (taken) into the indexed local history register. This history value is then used as the index for a separate 1K sized table holding 3-bit saturating counters for that local history. This 3-bit saturating counter is explained by the state-machine diagram pictured below in Figure 2.

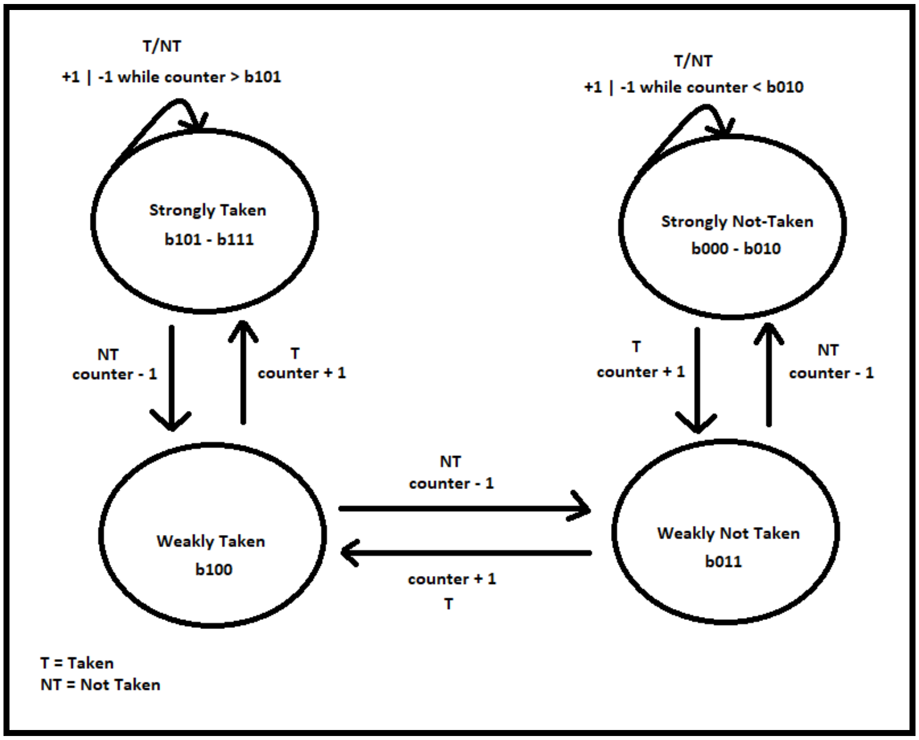

Figure 2: 3-bit Saturating Counter State Diagram

The second unit that must be constructed is the global history predictor. This is done by creating a 4k sized global predictor table (GPT) that holds 2-bit saturating predictors that function the same way as Figure 2, except it uses 2 bits instead of 3. The total bits that can be used to index the GPT is $log_2(4096) = 12$bits. The 12 bit index is a global history register that keeps track of the taken/not-taken status of the previous 12 branches the same way that the local branch history is done, by shifting a 1 or 0 bit into the register depending on if the previous branch was taken or not. The global predictor keeps track of the previous 12 branches regardless if they are the same branch or not, this is what makes this history register "global". The prediction value is taken from the MSB of the 2-bit counter, and the unit can be viewed below in Figure 3.

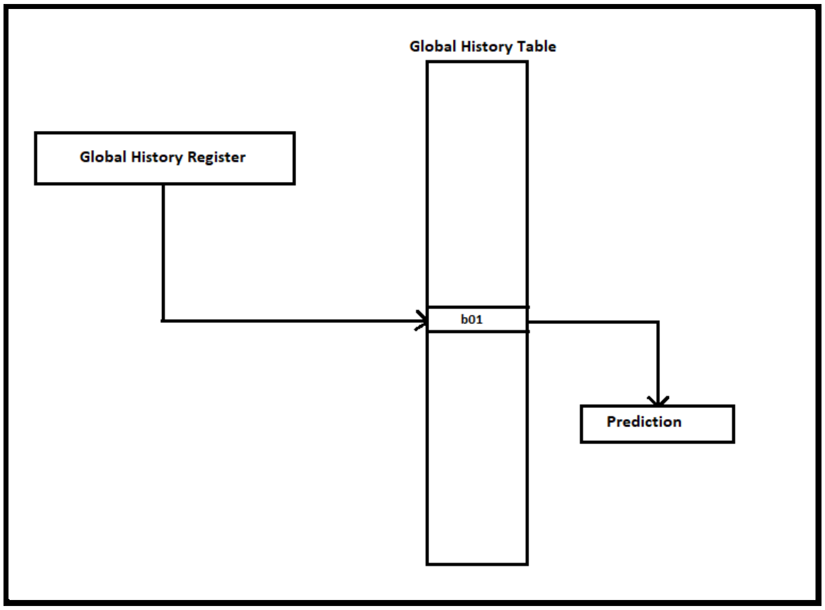

Figure 3: Global History Predictor

The final unit to be constructed to complete the Alpha21264 processor is to build a tournament predictor that dynamically chooses between the local and global history predictors. This is accomplished using the global history register as an index for a separate 4k table holding 2-bit saturating counters exactly like the global predictor in Figure 3. The MSB of the 2-bit counter value in this case is to choose the predictor instead of the branch prediction. A 0 chooses the local history predictor, and a 1 chooses the global history predictor. A full diagram of all the Alpha 21264 is shown in Figure 4.

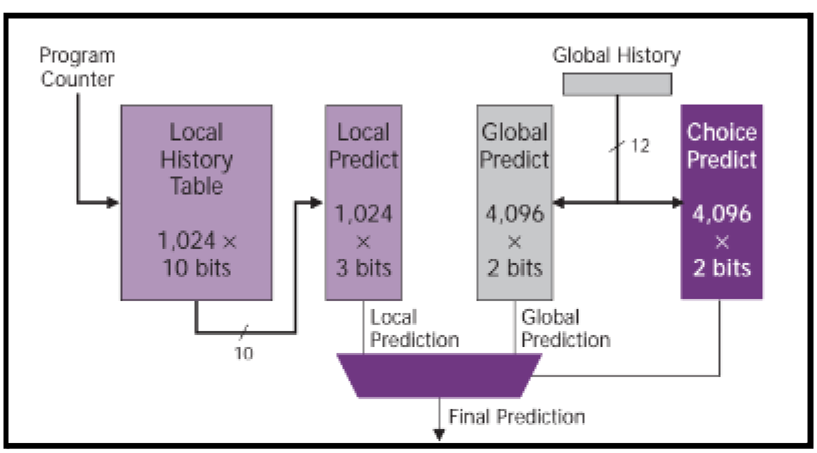

Figure 4: Alpha 21264 [1, Pg. 3]

These predictors with their corresponding registers and tables are saved to global registers within the ADL script to be updated as necessary by the EX stage of the pipeline. The ADL implementation of the predictors and the updated EX stage configuration can be found at the end of this document in the Appendix.

The new architecture is then compiled and run through the Spec95 INTEGER benchmark suite [5] included in the ADL/FAST simulation files using the “small” and “test” datasets. The architecture is also separately compiled two more times, benchmarking the local predictor and global predictor for comparison against the Alpha 21264’s CPI and conditional branch prediction performance.

RESULTS

Table 1 contains results from the “small” Spec95 suite benchmark test for the Alpha 21264.

Benchmark	Predicted Conditional Branches	% of Total Conditional Branches Predicted	CPI
099.go	4276157	49.08	1.344954
124.m88ksim	35359005	45.57	1.532819
130.li	79940435	51.47	1.487625
147.vortex	6953766	51.77	1.333571
126.gcc	178003	54.89	1.448677
129.compress	247442	77.50	1.143033
132.ijpeg	689749	53.82	1.352514
134.perl	755780	53.36	1.426711

Table 1: Alpha 21246 “Small” Benchmark Results

Table 2 contains results from the “small” Spec95 suite benchmark test for the Local Predictor.

Benchmark	Predicted Conditional Branches	% of Total Conditional Branches Predicted	CPI
099.go	4001720	45.93	1.341218
124.m88ksim	26267799	33.85	1.540792
130.li	67911646	43.72	1.512179
147.vortex	5322202	39.62	1.344799
126.gcc	129501	39.94	1.464955
129.compress	79313	24.84	1.257554
132.ijpeg	568789	44.38	1.373352
134.perl	615908	43.49	1.441858

Table 2: Local Predictor “Small” Benchmark Results

Table 3 contains results from the “small” Spec95 suite benchmark test for the Global Predictor.

Benchmark	Predicted Conditional Branches	% of Total Conditional Branches Predicted	CPI
099.go	4424419	50.79	1.338717
124.m88ksim	53132869	68.47	1.406703
130.li	71987558	46.35	1.511802
147.vortex	6070026	45.19	1.360377
126.gcc	157037	48.43	1.473372
129.compress	90410	28.32	1.268552
132.ijpeg	631859	49.30	1.370160
134.perl	684629	48.34	1.443274

Table 3: Global Predictor “Small” Benchmark Results

Figures 5 and 6 display the conditional branches successfully taken.

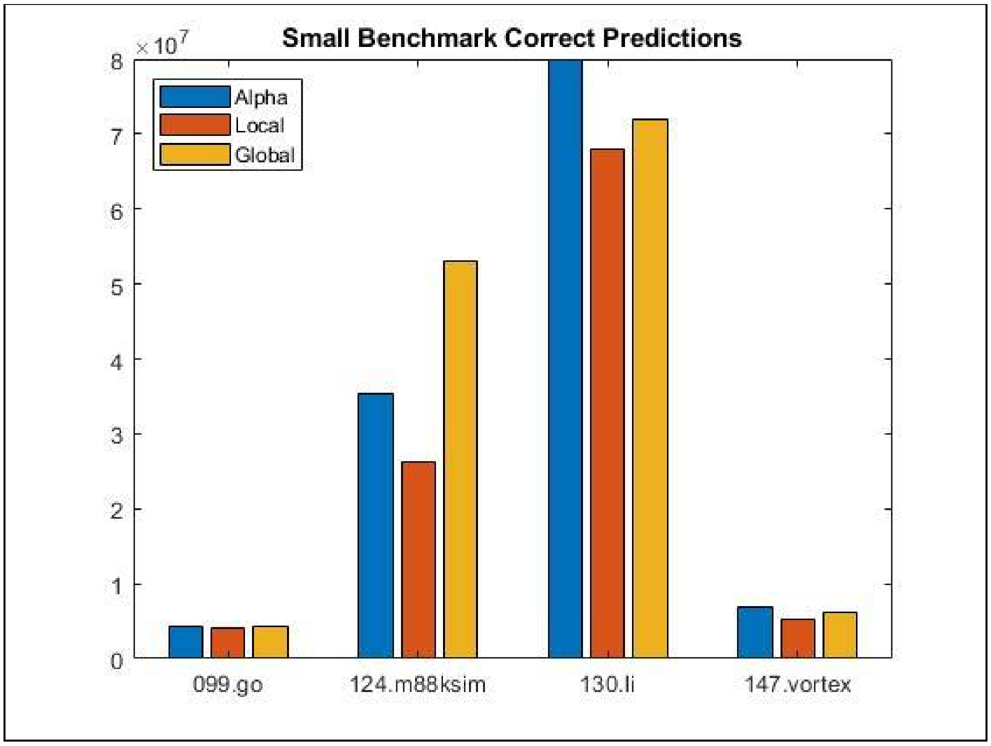

Figure 5: “Small” Benchmark Branch Prediction Results 1

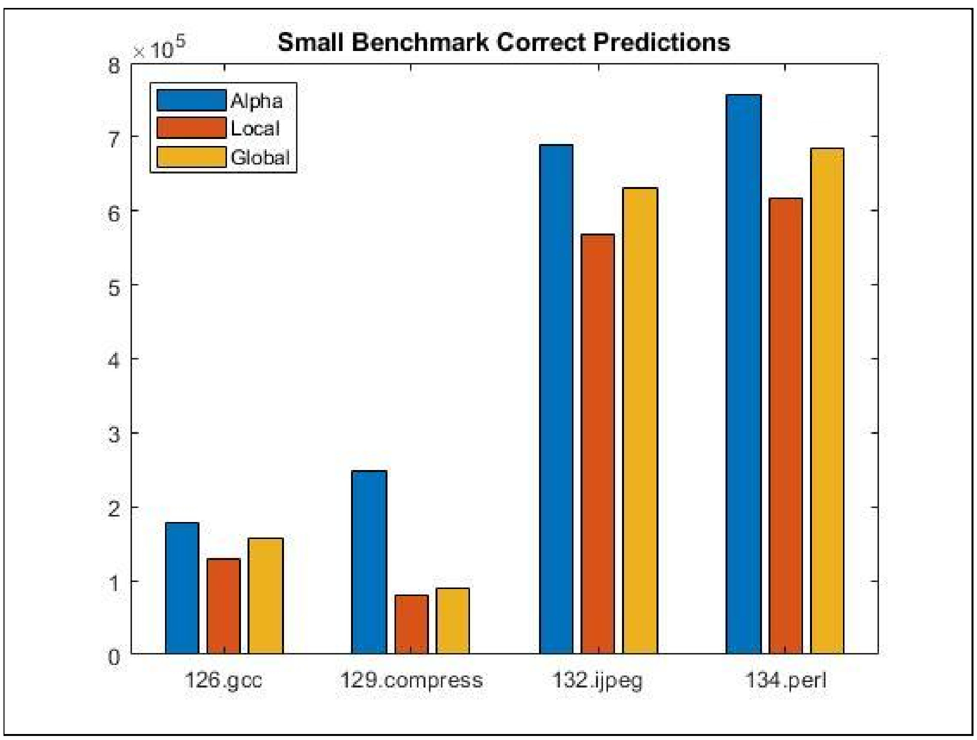

Figure 6: “Small” Benchmark Branch Prediction Results 2

Table 4 contains results from the “test” Spec95 suite benchmark test for the Alpha 21264.

Benchmark	Predicted Conditional Branches	% of Total Conditional Branches Predicted	CPI
099.go	861867188	48.50	1.313041
124.m88ksim	35359005	45.57	1.532819
130.li	79940435	51.47	1.487625
147.vortex	380851827	47.48	1.339420
126.gcc	5538223	59.04	1.457495
129.compress	16486758	48.22	1.261857
132.ijpeg	689749	53.82	1.352514
134.perl	755780	53.36	1.426711

Table 4: Alpha 21264 “Test” Benchmark Results

Table 5 contains results from the “test” Spec95 suite benchmark test for the Local Predictor.

Benchmark	Predicted Conditional Branches	% of Total Conditional Branches Predicted	CPI
099.go	822092076	46.27	1.306642
124.m88ksim	26267799	33.85	1.540792
130.li	67911646	43.72	1.512179
147.vortex	371018640	46.25	1.323538
126.gcc	3780552	40.30	1.485772
129.compress	15557779	45.51	1.256103
132.ijpeg	568789	44.38	1.373352
134.perl	615908	43.49	1.441858

Table 5: Local Predictor “Test” Benchmark Results

Table 6 contains results from the “test” Spec95 suite benchmark test for the Global Predictor.

Benchmark	Predicted Conditional Branches	% of Total Conditional Branches Predicted	CPI
099.go	904210177	50.89	1.304974
124.m88ksim	53132869	68.47	1.406703
130.li	71987558	46.35	1.511802
147.vortex	423693285	52.82	1.321145
126.gcc	4730013	50.42	1.493154
129.compress	18162512	53.12	1.247109
132.ijpeg	631859	49.30	1.370160
134.perl	684629	48.34	1.443274

Table 6: Global Predictor “Test” Benchmark Results

Figures 7 and 8 display the conditional branches successfully taken.

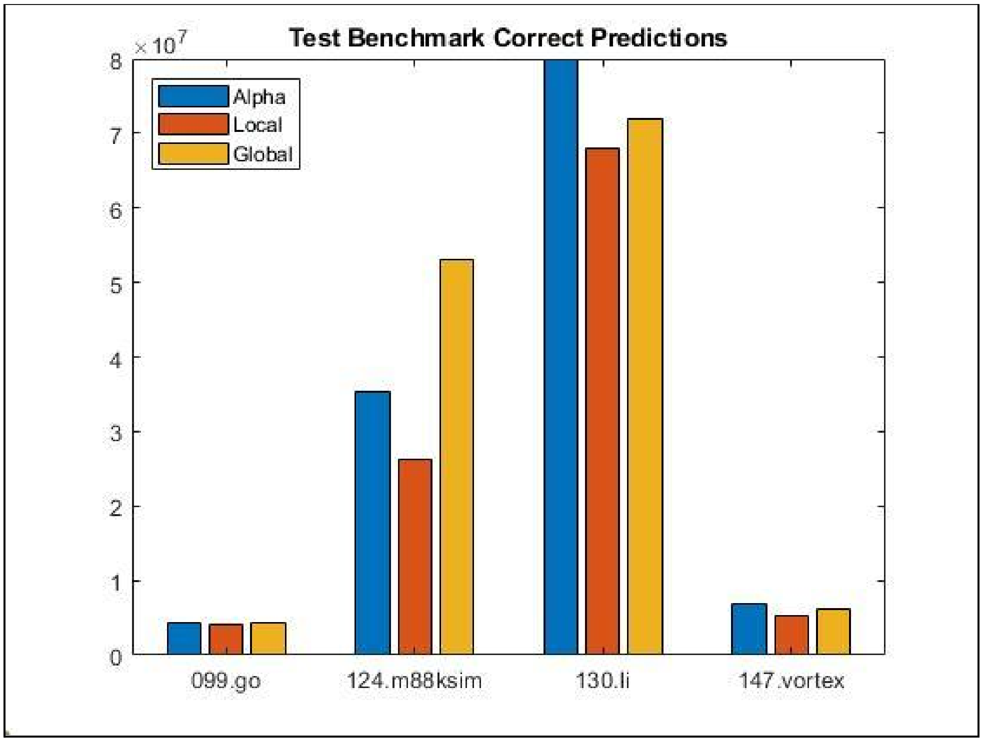

Figure 7: “Test” Benchmark Branch Prediction Results 1

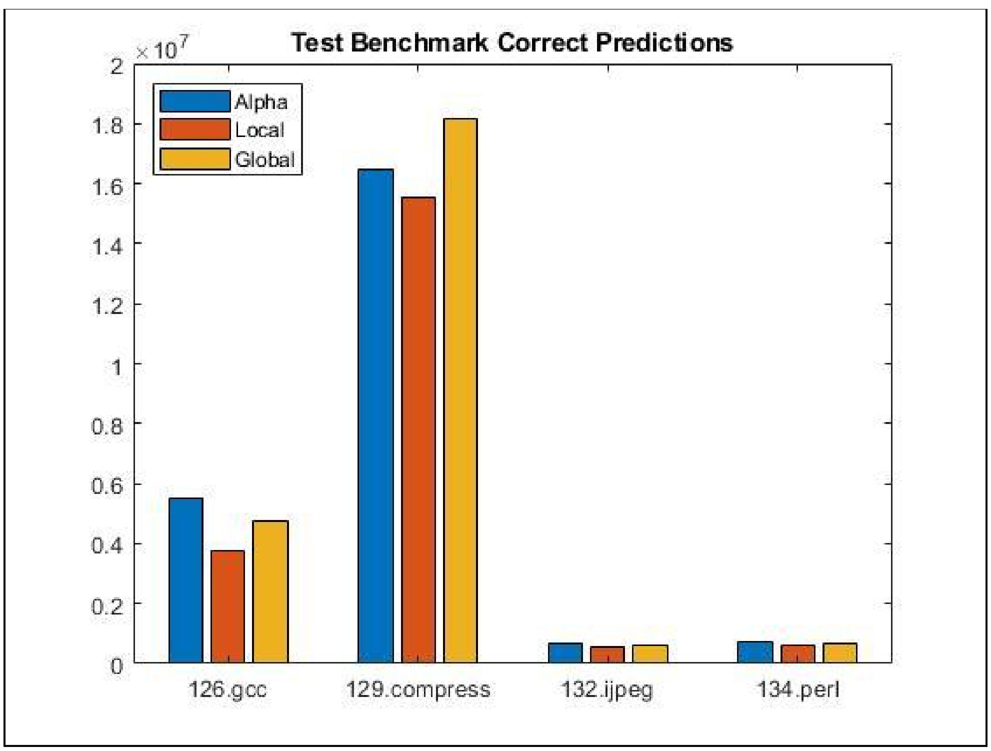

Figure 8: “Test” Benchmark Branch Prediction Results 1

Figures 9 and 10 display the CPI of each benchmark. (Lower CPI is better.)

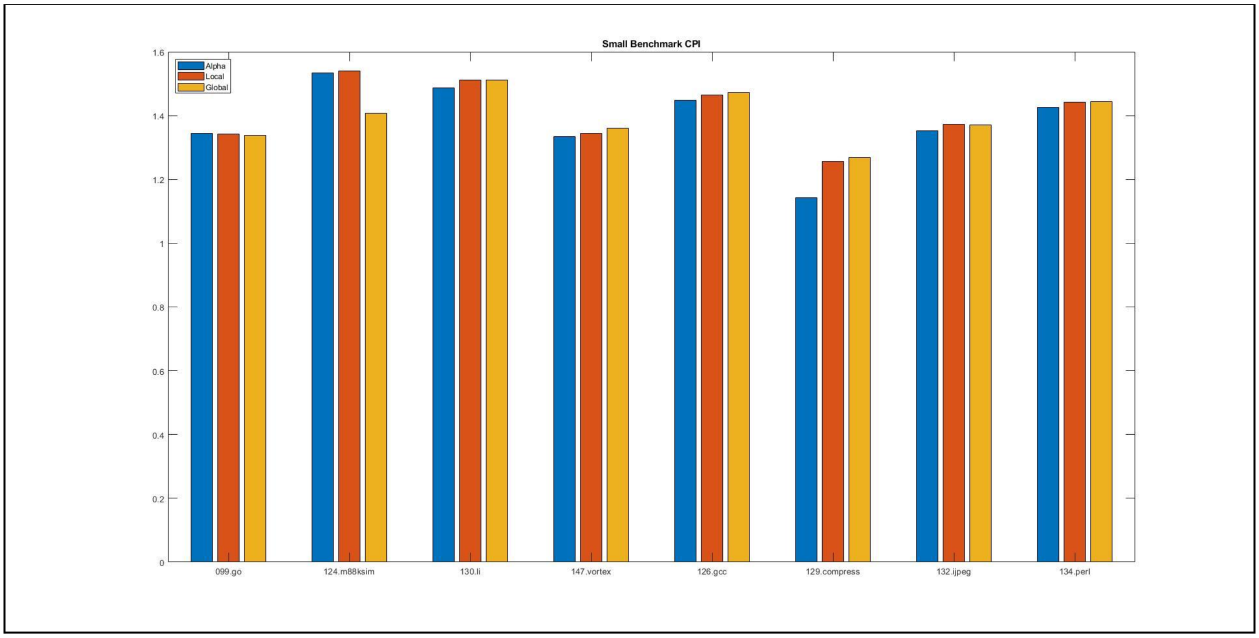

Figure 9: “Small” Benchmark Branch Prediction Results 2

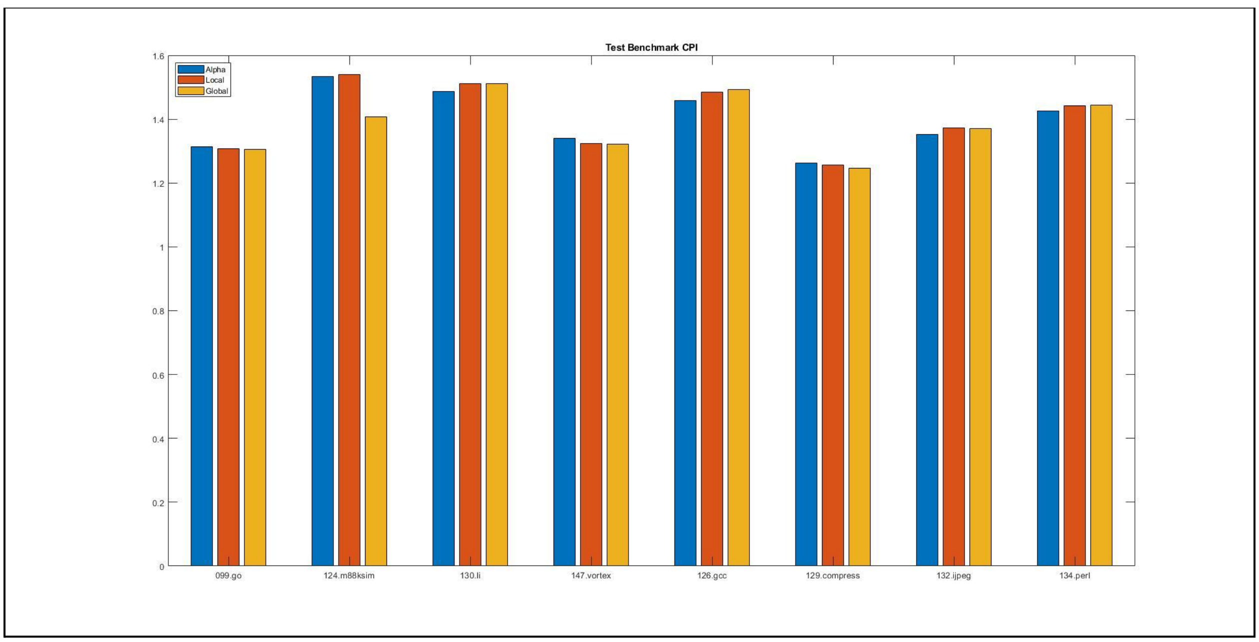

Figure 10: “Small” Benchmark Branch Prediction Results 2

Predictor	Average % of Predicted Conditional Branches	Average CPI
Alpha 21264	52.81	1.390
Local	41.22	1.407
Global	50.31	1.392

Table 7: Averages

Viewing the results of the small benchmarking test, it can be seen in Figures 5 and 6 with Tables 1 through 3 that the Alpha 21264 had predicted the conditional branch in ¾ of the tests better than using the stand-alone global or local predictors. The Alpha only lost slightly to the global predictor in the “124.m88ksim” benchmark, and all three were about even during the “099.go” benchmark. The CPI data viewed in Figure 9 backs up the conditional branching data, showing that the CPI of the Alpha was higher than the global predator’s CPI during the “124.m88ksim” and “099.go” benchmarks.

During the larger “test” benchmark, it can be seen again that in Figures 7 and 8 with Tables 4 through 6 that the Alpha 21264 had predicted more conditional branches than the stand-alone global or local predictors. This time though it had lost to the global predictor in the “129.compress” benchmark as well. It is interesting to note that a lower CPI did not necessarily correlate with predicting more conditional branches as seen in the comparison of the Alpha and the local predictor in the“147.vortex” benchmark.

Looking at Table 7, though, the Alpha 21264 predicts conditional branches and has a lower CPI on average compared to the two separate predictors. The Alpha guessed 2.5% more conditional branches than the global predictor, and 11.59% more branches than the local predictor. The Alpha also had slightly less CPI than the global predictor, and a 0.017 CPI lower than the local predictor.

CONCLUSION

The Alpha 21264 performed better than the local or global predictors independently according to the results gathered from the Spec95 benchmark for the “small” and “test” data sets. It was noted that the Alpha predictor lost slightly to the global predictor on certain individual benchmarks. The slowness caused by the Alpha 21264 in these instances could be because of the choice predictor taking more time in choosing the less-efficient local predictor before switching. More research and experimentation is needed to be done on the matter.

In conclusion, the Alpha 21264 predictor predicted more conditional branches and has a lower CPI on average, upholding the theory stated in the beginning.

REFERENCES

1. L. Swennap, “Digital 21264 Sets New Standard; Clock Speed, Complexity, Performance Surpass Records, But Still a Year Away”, *Microprocessor Design,* Microprocessor Report, Oct. 28, 1996, [Online] Available: https://studies.ac.upc.edu/ETSETB/SEGPAR/microprocessors/21264%20(mpr).pdf, [Accessed: Dec. 12, 2020]
2. “Ubuntu 20.04.1 LTS”, Ubuntu, 2020, [Online] Available: https://ubuntu.com/download/desktop, [Accessed: Dec. 12, 2020]
3. S. Onder, “An introduction to Flexible Architecture Simulation Tool (FAST) and Architecture Description Language ADL”, *Michigan Technological University,* Houghton Michigan, United States, 2020, [Online] Available: https://pages.mtu.edu/~soner/fast_manual.pdf, [Accessed: Dec. 12, 2020]
4. S. McFarling, “Combining Branch Predictors”, *Western Research Laboratory*, WRL Technical Note TN-36, Palo Alto, California, United States, Jun. 1993, [Online] Available: https://www.hpl.hp.com/techreports/Compaq-DEC/WRL-TN-36.pdf, [Accessed: Dec. 12, 2020]
5. “SPEC95 CPU95 BENCHMARKS”, Standard Performance Evaluation Corporation, Sep. 26, 2003, [Online] Available: http://www.spec.org/cpu95/, [Accessed: Dec. 12, 2020]

APPENDIX

```
# Init arrays for predictors and choice predictor
branch_target_buffer file btb [btb_entries];
integer array lht[lh_size, 10];
integer array lhb[lh_size, 3];
integer array gh[1, gh_size];
integer array gpt[gp_size, 2];
integer array cp[cp_size, 2];
```

Appendix 1: alpha.adl Predictor Table Initializations

```
constant
     lh_size                  1040,
     gh_size                    12,
     gp_size                  4096,
     cp_size                  4096,
     btb_entries            131072,     #- In ent
     branch_global_bits         14,     #- Gshare

     branch_predictor          1,       #- Equate
     gshare_predictor          1,
     counter_initial_value     2,
     counter_lh_init_value     4,
     counter_pr_init_value     2,
     counter_max_value         3,
     counter_lh_max_value      7,
     counter_gh_max_value      3,
     counter_cp_max_value      3,

     #
     lht_entries               1024,
     lh_mask                   lh_size - 1,
     gp_mask                   gp_size - 1,
     cp_mask                   cp_size - 1,
     branch_target_mask        btb_entries - 1;
```

Appendix 2: alpha.adl Constant Initializations

```
controldata register
   my_pc          32,
   branch_target  32,
   my_btb_index   32,
   my_lht_index   32,
   my_lh_index    32,
   my_gpt_index   32,
   my_cp_index    32,
   my_pre_index   32,
   btb_hit         1,
   btb_type        6,
   prediction      1,
   predict_target 32,
```

Appendix 3: alpha.adl Control Data Register Initializations

```
# Init gloabl integers
integer
   gpt_index,
   lht_index,
   lh_index,
   cp_index,
   branch_gr,
   rb_branch_gr;
```

Appendix 4: alpha.adl Global Integer Initialization

```
procedure predict_taken_not_taken untyped [branch_predictor == gshare_predictor]
begin
        # Init local variables to be used for prediction
        local counter_value;
        local counter_predictor;
        local target_addr;
        local btb_index;
        local predict;
        local predictor_choice;

        # Check for btb hit
        if btb_hit then
        begin
                # Get target address and local btb
                btb_index = my_btb_index;
                new_pc = btb[btb_index].b1_target_address;

                # Get choice predictor
                cp_index = gh[0] & cp_mask;
                my_cp_index = cp_index;
                counter_value = cp[cp_index];
                predictor_choice = counter_value.[1:1];
```

Appendix 5: alpha.adl IF Stage Predictor Implementations Part 1

```
        # Determine branch type
        case btb[btb_index].b1_branch_type of
        begin
                conditional_direct          :
                conditional_direct_link     :
                        # Decide which predictor to use
                        if predictor_choice then
                        # Use Local predictor if choice == 1
                        begin
                                lht_index = my_pc.[9:0] & lh_mask;
                                my_lht_index = lht_index;
                                lh_index = lht[lht_index].[9:0] & lh_mask;
                                my_lh_index = lh_index;
                                counter_value = lhb[lh_index];
                                predict = counter_value.[2:2];
                        end
                        # Else Use Global predictor of choice == 0
                        else
                        begin
                                gpt_index = gh[0].[11:0] & gp_mask;
                                my_gpt_index = gpt_index;
                                counter_value = gpt[gpt_index];
                                predict = counter_value.[1:1];
                        end;
                unconditional_indirect      :
                unconditional_indirect_link :
                unconditional_direct_link   :
                        predict=true;
                        new_pc = btb[btb_index].b1_target_address;

                unconditional_direct        :
                        predict = true;
                        new_pc = btb[btb_index].b1_target_address;
        end;

        # If there is no prediction increase counter
        if predict == 0 then
                new_pc = my_pc + 4;
end
```

Appendix 6: alpha.adl IF Stage Predictor Implementations Part 2

```
        # No branch was detected
        else
        begin
                new_pc = my_pc + 4;
        end;

        # Set global predictor and pc values
        predict_target = new_pc;
        prediction = predict;

end predict_taken_not_taken [branch_predictor == gshare_predictor];
```

Appendix 7: alpha.adl IF Stage Predictor Implementations Part 3

```
#- Instruction Execution  - - - - - - - - - - - - - - - - - - - - - - - -#
#                                                                         #
#                                                                         #
#- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #

procedure s_EX prologue
begin
   local next_pc;
   local mispredict;
   local counter_predictor;
   local choice_predictor;

   do_forwarding_to_execute;

   if is_branch then
      begin
         if (c_what ^= condition_z) then
             branch_target=my_pc + sign_extend_14(literal);

         if i_class == float_class then
            branch_input=lop == tf          # gpr[$CpC+34] == tf
         else
            condition_code(lop,rop);

         taken = branch_input;

         if branch_input then
            next_pc = branch_target
         else
            next_pc = my_pc + 4;

         #- update BTB & PHT
         btb[my_btb_index].b1_branch_type=ordinal(c_detail);
         btb[my_btb_index].my_address = my_pc;
         btb[my_btb_index].b1_target_address=branch_target;
```
Appendix 8: alpha.adl EX Stage Predictor Configuration Part 1

```
# Get predictor index
counter_predictor = gh[0];
choice_predictor = counter_predictor.[1:1];

# Check branch condition
case c_detail of
begin
       conditional_direct          :
       conditional_direct_link     :
               # Check if prediction was correct
               if prediction == branch_input then
               # Prediction is correct
               begin
                       # Check which branch predictor was used
                       if choice_predictor then
                       # Local predictor was used
                       begin
                               # Increase the local branch history saturated counter
                               lht_index = my_lht_index;
                               lh_index = my_lh_index;
                               lht[lht_index] = (lht[lht_index] << 1 | 1) & lh_mask;
                               if lhb[lh_index] < counter_lh_max_value then
                                       lhb[lh_index] = lhb[lh_index] + 1;

                       end
                       # Global predictor was used
                       else
                       begin
                               # Increase the global prediction counter
                               gpt_index = my_gpt_index;
                               if gpt[gpt_index] < counter_gh_max_value then
                                       gpt[gpt_index] = gpt[gpt_index] + 1;
                       end;

                       # Reflect global history changes
                       gh[0] = ((gh[0] << 1) | 1);
                       correct_predictions = correct_predictions + 1;
               end
```

Appendix 9: alpha.adl EX Stage Predictor Configuration Part 2

```
                        else
                        # Prediction was incorrect
                        begin
                                # Check which branch predictor was used
                                if choice_predictor then
                                # Local predictor was used
                                begin
                                        # Decrease the local branch history saturated counter
                                        lht_index = my_lht_index;
                                        lh_index = my_lh_index;
                                        lht[lht_index] = (lht[lht_index] << 1) & lh_mask;
                                        if lhb[lh_index] then
                                                lhb[lht_index] = lhb[lht_index] - 1;
                                end
                                # Global predictor was used
                                else
                                begin
                                        # Decrease the global prediction counter
                                        gpt_index = my_gpt_index;
                                        if gpt[gpt_index] then
                                                gpt[gpt_index] = gpt[gpt_index] - 1;
                                end;

                                # Reflect Global history changes
                                gh[0] = (gh[0] << 1) & gp_mask;
                        end;

                        conditional_branches = conditional_branches + 1;
         end;

         if (btb_hit == 0) | (prediction ^= branch_input) then
            branch_gr=rb_branch_gr;

         if next_pc ^= predict_target then
            begin
               mispredict = true;
               new_target = next_pc;
               if has_context s_RF then squashed[s_RF] = 1;
               if has_context s_ID then squashed[s_ID] = 1;
               if has_context s_IF then squashed[s_IF] = 1;
            end
         else
            mispredict = false;
        end;
end s_EX;
```

Appendix 10: alpha.adl EX Stage Predictor Configuration Part 3