

# Single Core Design Space Exploration

ECE 568: Advanced Microprocessor Architecture, Spring 2016

Anthony Gallotta  
agallo4@uic.edu

## ABSTRACT

This paper seeks to find the best single core superscalar architecture for a given benchmark application by experimenting with different architectures for branch prediction, memory systems, functional units, data path, and other areas. The metrics used to define the "best" architecture are *performance* as instructions per cycle (IPC) and *performance-energy* as the energy delay product (EDP).

## 1. INTRODUCTION

This study evaluates a single core superscalar architecture using the SimpleScalar simulator suite, seeking the best configuration for the **eeg** benchmark application. A base architecture was given, and design modifications were then made in four groups, branch prediction, memory system, functional units, and data path. In the first phase of design space exploration, tuning one parameter at a time. In the second stage, closely related parameters within the same groups are tuned together. Finally, the best configurations across groups are combined to formulate the best found architecture. For each configuration we observe performance (IPC) and performance-energy (EDP), calculated as  $(CPI * energy/cycle) * CPI$ . The "best" configuration seeks to maximize performance, and minimize performance-energy.

## 2. SIMULATION

Simulations of the **eeg** application benchmark were performed using **sim-outorder**. The base configuration had performance of 1.5226 IPC, and a performance-energy product of 219.68. While many parameter changes were tested, unless otherwise noted, all graphs that follow show only configurations that outperformed the base configuration in at least one of these metrics<sup>1</sup>. The most desirable configura-

<sup>1</sup>Results from all configurations, along with the configuration files themselves and scripts used in this simulation are available at <https://github.com/tonygallotta/ece568-single-core-dse.git>

Table 1: Instruction Profile for **eeg**

Type	Count	Percent
load	171,182,879	20.30
store	84,069,724	9.97
uncond branch	60,740,822	7.20
cond branch	59,922,458	7.11
int computation	335,861,146	39.84
fp computation	131,306,086	15.57
trap	1,035	0.00

tions will lie in the lower right side of the graphs that follow.

To gain some direction in performance tuning, the instruction profile for the benchmark was first evaluated using **sim-profile**. As shown in table 1 Instruction Profile for **eeg**table.1, the benchmark is primarily an integer program. Given this instruction breakdown, most of the performance tuning was focused on branch prediction and the memory system. For each group, the initial focus was on one attribute at a time. Once good parameters to tune were found, some combinations of parameters within each group were attempted as well to further improve performance.

### 2.1 Branch Prediction

The first branch prediction change attempted was to use static prediction. This strategy resulted in much worse performance by both metrics and was quickly abandoned. Next, some tuning was made with the default bimodal branch predictor, which uses a simple strategy of picking the most common direction. Sticking with this scheme, the branch target buffer (BTB) size and associativity was adjusted. In figure 1 Branch Predictor Variation resultsfigure.1, the points labeled as **bp-btb-<sets>-<associativity>** indicate the results of these adjustments. The BTB configuration of 128 sets with 2 way associativity is able to achieve the same performance as the base configuration, with a much lower EDP.

Two level branch predictors use a combination of the last  $k$  historical branch outcomes, as well as the behavior for a specific pattern of previous branches [1]. While many of the 2 level configurations did not perform the base configuration, the 2 level Global Address (GA) branch predictor with a 4 wide history (bp-2level-hist-4) did achieve a better IPC. Combining these configuration changes (bp-comb-128-2) by keeping the best found BTB size and using a combined bimodal and 2-level branch predictor gives about a 5% performance-energy improvement over the base configuration.

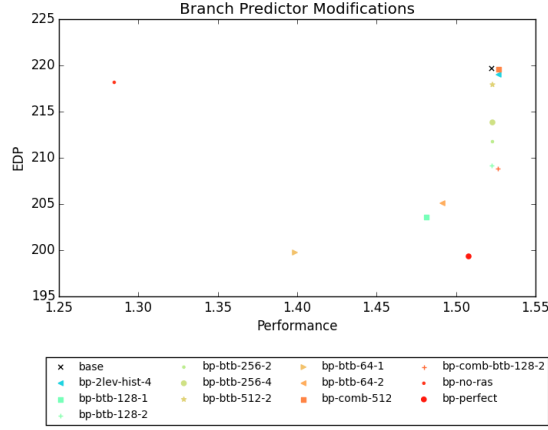


Figure 1: Branch Predictor Variation results

A perfect branch prediction scheme is plotted for comparison. Surprisingly, the perfect branch predictor achieves a lower IPC than many of the experimental configurations. The simulator documentation does not describe in detail how the perfect branch predictor operates, so it may be that they are modeling a higher latency for the perfect predictor.

## 2.2 Memory System

The simulator has a 2 level cache hierarchy, and we experiment with different sizes, associativity, block sizes, and replacement policies at each level, and for both instruction and data caches.

### 2.2.1 L1 Cache

As figure 2 Modifications for L1 data cache figure.2 shows, increasing the size of the L1 data cache (sets) did not result in significant IPC improvements relative to their increased power consumption. Increasing the block size to 64 while keeping the total L1 size the same (bsize-64) yields the best result in terms of both of our architecture goals.

Figure 3 Modifications for L1 instruction cache figure.3 shows similar results for the instruction cache - a block size of 64 performs best.

### 2.2.2 L2 Cache

At the L2 cache, we see once again from figures 4 Modifications for L2 data cache figure.4, 5 Modifications for L2 instruction cache figure.5, and 8 Data path modifications figure.8 that a unified cache with a decreased block size performs best.

## 2.3 Functional Units

Since this program is integer focused, one of the first modifications made to increase performance-energy was to decrease the number of floating point units. As figure 7 Functional unit modifications figure.7 shows (fpalu-2), the number of floating point ALUs can be decreased from 4 to 2 with no change in IPC. Interestingly, increasing the number of FP multipliers does increase IPC, at a slight cost to EDP.

The integer functional units seem to already be at a sweet spot. Increasing the number of integer ALUs or multipliers does not result in significant IPC gains. The base configuration only has 1 integer multiplier, so this cannot be

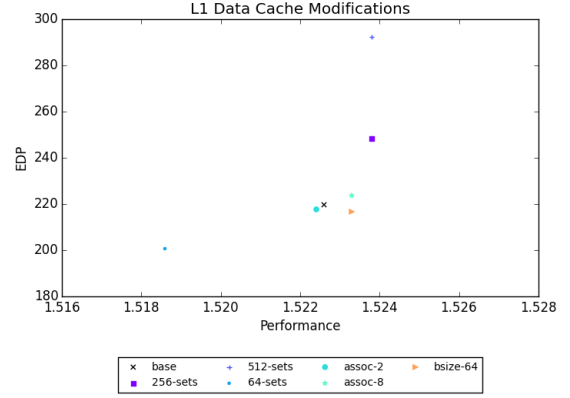


Figure 2: Modifications for L1 data cache

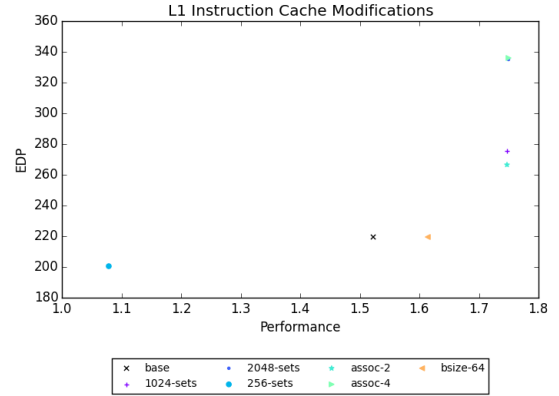


Figure 3: Modifications for L1 instruction cache

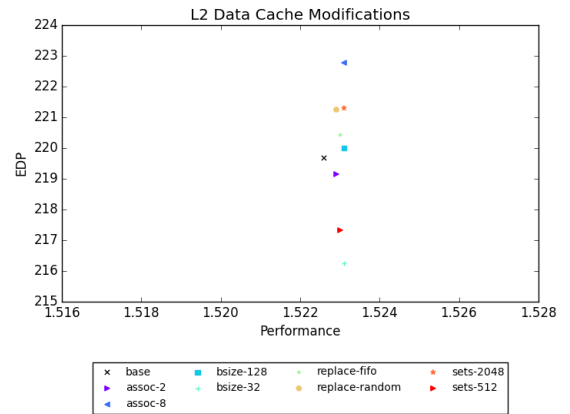


Figure 4: Modifications for L2 data cache

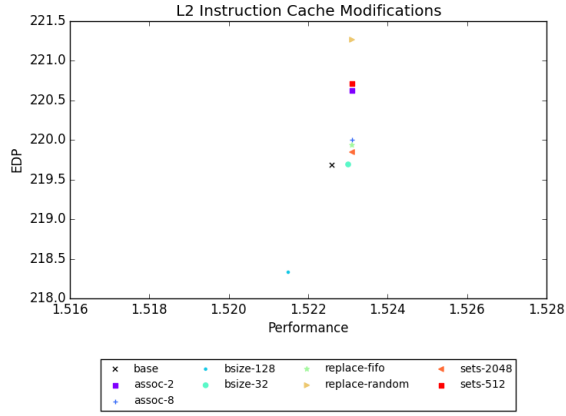


Figure 5: Modifications for L2 instruction cache

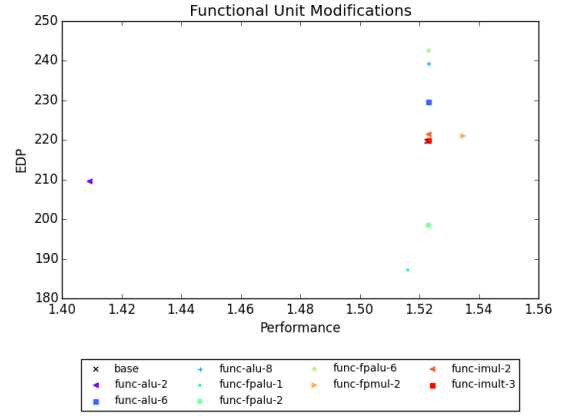


Figure 7: Functional unit modifications

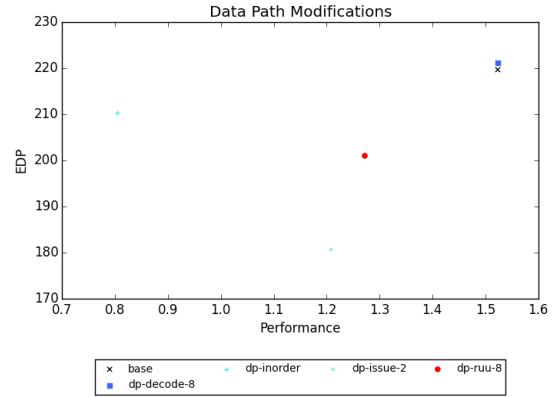


Figure 8: Data path modifications

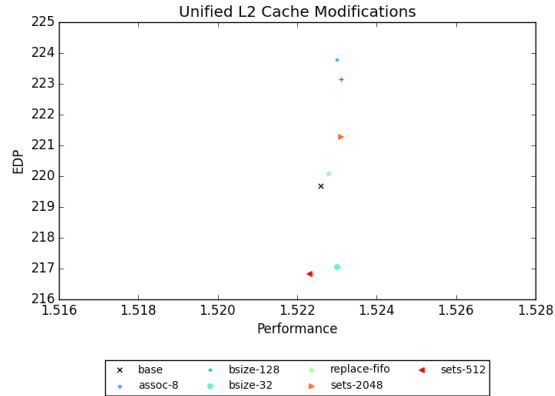


Figure 6: Unified L2 cache

decreased, but decreasing the number of ALUs results in significant IPC losses.

## 2.4 Data Path

The data path attributes modified were in-order vs. out of order (base) issue, instruction decode and issue width, and size of the reorder. On their own, none of these modifications result in a good trade off of performance vs. EDP. These attributes will be revisited when tuning parameters are combined since other modifications may make a wider instruction issue or reorder buffer more useful.

## 3. CROSS GROUP SIMULATIONS

## 4. CONCLUSIONS

This paragraph will end the body of this sample document. Remember that you might still have Acknowledgments or Appendices; brief samples of these follow. There is still the Bibliography to deal with; and we will make a disclaimer about that here: with the exception of the reference to the L<sup>A</sup>T<sub>E</sub>X book, the citations in this paper are to articles which have nothing to do with the present subject and are

used as examples only.

## **5. REFERENCES**

- [1] T.-Y. Yeh and Y. N. Patt. A comparison of dynamic branch predictors that use two levels of branch history. In *ACM SIGARCH Computer Architecture News*, volume 21, pages 257–266. ACM, 1993.