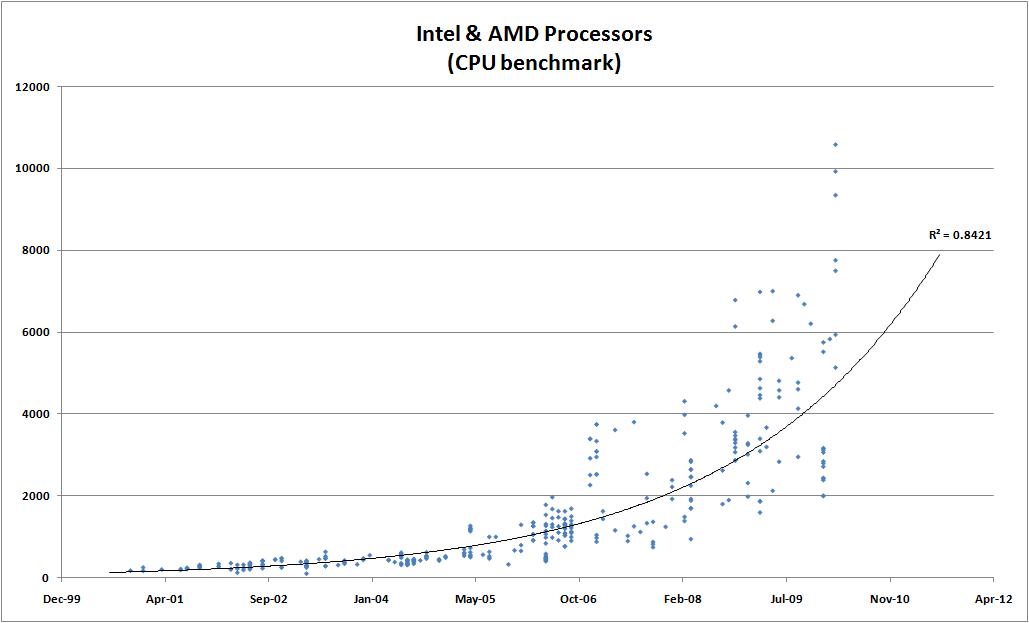
## CMPEN 431

Simple Scalar Final Project



URL: http://www.glohse.com/wp-content/uploads/2010/05/CPU\_Performance-300x181.jpg

Written By: Erik Arfvidson

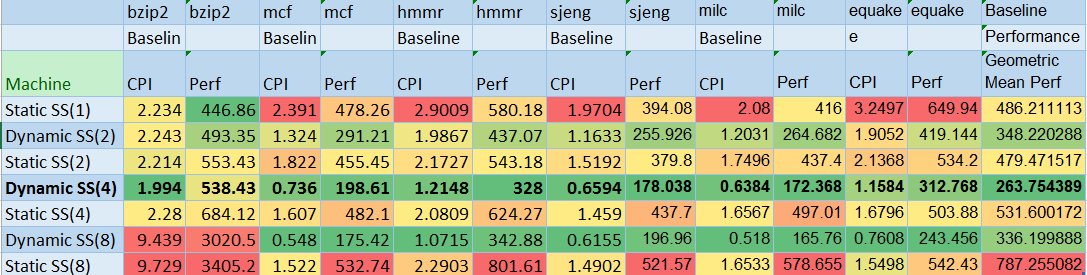
# Introduction

The purpose of this project is to work with the SIMPLE SCALAR environment to create an optimized configuration file to achieve the “best” design. The file must have certain specific constraints, with the rest open to the discretion of the tester. My main goal for this project is to design an optimal CFG file using bash scripts to speed up the data gathering process. I will also conduct several experiments simultaneously and compare the results to the baseline configuration results in order to determine if my new settings have improved or hindered the performance. The configuration file will be simulated using six benchmark and then adding up the results.

The Setup that I used to conduct my experiment was the UNIX environment provided by lab218 which I either connected to locally or remotely using a VPN client. I also wrote several bash scripts before initiating the project because I wanted to streamline the testing process. The features of my bash scripts were to quickly conduct multiple sets of simulation and retrieve the CPI information.

# Experiment 1

The first set of experiments that I decided to simulate as my initial starting ground for this project were to determine if Static or Dynamic with a given ifqsize would give the highest geometric performance. In order to modify the ifqsize, you also are required to change the issue width and decode width setting.



**Figure 1.1**: Table: Results of experiment 1 simulation on ifqsize

In Figure 1.1 we can easily spot the dynamic SS 4 way to have the best geometric performance of 263.75 given that the CPI for the six benchmarks remained low. This result although interesting was not as surprising since I have been working on it before in the previous homework assignments.

**Figure 1.2**: Graph: Performance of the six benchmarks (ifqsize)

As you can see, in this Figure 1.2 the yellow bar’s remained the smallest throughout the tests which clearly denotes that Dynamic SS(4) had the best performance. Also notice that the 4way and 8way performance for both static preformed negatively in the bzip2 simulation resulting in the slowest performance on the geometric graph as you can see in Figure 1.3.

**Figure 1.3:** Graph: Geometric Performance of the six benchmarks ifqsize(Small is Better)

Figure 1.3 clearly illustrates the winner of experiment Dynamic SS(4) significantly outperformed the others. To conduct the following experiments I used Dynamic SS(4) as my default configurations.

# Experiment 2

In Experiment2, I used the results of experiment 1 as my new baseline to conduct my test. Now that I had determined the width of my CPU, I decided to test to see which configuration of the il1 cache size would output the optimal CPI.

**Figure 2.1:** Graph: Geometric Performance of four 2way il1 settings

Figure2.1 data demonstrated a huge performance boost from the default configuration which output a geometric performance for248.38(128:32:02) to the best performance which was 222.58(256:32:02). The 256 il1 had a 5.47% performance boost which was a significant improvement to the benchmark.

**Figure 2.2:** Graph: Geometric Performance of the four configuration 4 way il1 settings (Small is Better)

In this set of test I also noticed a significant improvement when using the 128:32:04 however 222.58(256:32:02) configuration had a better overall performance.

**Figure 2.3:** Graph: Geometric Performance of all the il1 setting configuration.

Figure 2.3, Demonstrates that the best performance with different il1 settings was 222.58(256:32:02).

# Experiment 3

After finding the optimal il1 cache for my 4 way dynamic SS. The next thing that I decided to test was the ul2 configurations which had a slight performance improvement with the base which was the previous one with no ul2 settings.

**Figure 3.1:** Graph: Geometric Performance of all the ul2 4 way settings

Figure 3.1, shows how the ul2 setting 2048:64:4 demonstrated to be the best configuration for a 4 way setting but as you can see from figure 2.2, it did not have a better geometric performance. The 8 way and 18 way configuration didn’t have a significant improvement, so I used ul2 setting 2048:64:4.

# Experiment 4

Experiment 4, was very appalling since I didn’t foresee that tlb would have any significant performance improvement with any of the configurations.

**Figure 4.1:** Graph: CPI tlb settings (Small is Better)

Figure 4.1, shows my results when using different tlb configuration that I got the same CPI results for all of them. Since there was no performance increase with bzip benchmark I decided to continue with my experiment.

# Experiment 5

The next setting that I experimented with and that also showed a significant performance improvement was the RUU setting.

**Figure 5.1:** Graph: CPI Performance of the six benchmarks with 7 different ruu settings

**Figure 5.2:** Graph: Geometric Performance of the seven ruu settings

Figure 5.1 and Figure 5.2, shows that ruu setting configuration of 64 and 128 are the best performing configuration. This might be expensive when considering cost, but since we are just concerned of performance, it definitely a performance booster.

# Experiment 6

In experiment 6 I performed test using different lsq settings on my cfg file.

**Figure 6.1:** Graph: CPI Performance of the six benchmarks lsq settings (Small is Better)

**Figure 6.2:** Graph: Geometric Performance lsq settings

Figure 6.1 and 6.2 Graphs, show the CPI and geometric performances on the different lsq settings. As you can observe again the highest setting is the best but might also be the most expensive in a real world scenario.

# Experiment 7

Here I conducted my final simulation. I first did some research on the remaining setting and what configuration would improve performance and then I conducted several tests. The final base setting configuration file I used as my best result of my lsq performance. Then I modify and add the memport, commit, mult, ruu, and lsq to their max setting.

**Figure 7.1:** Graph: Geometric Performance of my three last configuration file

My final simulation results were interesting because I got a negative result for my pre final file after adding the settings I researched online. After noticing the negative performances, I then tweaked my mem port to a higher setting. I also increased my multiplier to the highest setting. The setting that resulted in the highest geometric performances was the commit setting.

# Summary and Conclusions

The lab provided great learning experience since it expanded my basic knowledge on bash file and Linux environments. I achieve a performance of 148.83. I am still sure that a higher geometric performance could be achieved. If I had more time to continue testing, I would retest my tlb configuration with the new settings configuration that I optimized to see if I could get better results. I would also like to attempt this project with a brute force algorithm to see how high of a performance can be achieved with the initial constraints.