CMPEN 431 - Final Project

Quang Nguyen & Richard Heidorn

December 12, 2015

Best IPC and Execution Time for **mcf** and **milc**:

	IPC	Execution Time
Base mcf		
Base milc		
Best mcf		
Best milc		

Best execution time **mcf** issue width and data path type:

Best execution time **milc** issue width and data path type:

Overall Best Execution Time Geometric Means:

	Geometric Mean
Best Integer GM	
Best Floating Point GM	

Best execution time GM Integer issue width and data path type:

Best execution time GM Floating Point issue width and data path type:

1 Introduction

Designing a computer architecture is far from an exact science. A computer's performance is determined by many variables, ranging from the design and efficiency of the processor to the operating system and programs that run on top of the hardware. To name only a few, a computer's performance is determined by the functional units its processor contains, the length of its busses, the size and number of transistors, the organization of its caches, the speed and reliability of its hard drive, and ultimately the efficiency of its software.

By and large, computer architectures have become exponentially faster, smaller, more efficient, more durable, and more powerful. However, the number of factors that predict the performance of programs with the hardware is so great, it is nearly impossible to predict the most efficient computer designs for any given purpose. After all, the design that is optimal for one application might be sluggish for another. Even the same program, given a different set of data to compute, could perform better on different architectures.

For this project, we've investigated many different computer architectures using the Simple Scalar architecture simulator and how they affect the overall geometric means for four integer benchmarks and two floating point benchmarks provided by the SPEC performance benchmark package. The four integer benchmarks - bzip2, hmmer, mcf, sjeng - and two floating point benchmarks - milc, equake - were used to evaluate the overall performances of each tested architecture, and to evaluate how certain changes to the architectures would impact the programs' performance.

1.1 Testing Methodology

Simple Scalar provides a number of parameters that can be modified to emulate any modern computer architecture design. While the number and purpose of the parameters is easy to understand, the combinations of different parameters is incredibly large and difficult to comprehend. Thankfully, the project has been defined to limit the number of architectures that could be tested, but the sum total of all combinations is far greater than can be reasonably tested in a few weeks' time. Therefore, a brute force approach to determining the best design is impractical at best.

However, due to the number of variables outside of the architecture - the algorithms, access patterns, and behavior of the programs tested - it is not simply good enough to make educated guesses based on our understanding of processor evolution. Instead, a combination approach is required. Scientific reasoning is necessary to isolate the parameters and restrict the range of values which could benefit the performance of a given design. Once the parameters are defined and the range of reasonable values determined, a series of tests need to be run to experimentally verify our predictions and also to determine the best design decisions for a given architecture.

Our testing methods combine scientific analysis and experimental verification, both to determine which variables to test and which tests to run for all static and dynamic issue machines. Following each suite of tests, we identified and analyzed those designs which worked best for each machine and issue width. We've run dozens of test suites which isolated different components of the processor's functional units, branch prediction, instruction issuing, cache design, and the TLB. Each of these tests suites contained dozens to hundreds of individual configurations, all of which provided experimental information that were used to advance our designs to find a performant system. These tests were not comprehensive in any way, but we believe that we've isolated strong designs as a result of our methodology.

1.2 Calculating the Geometric Mean

The geometric means were calculated using the program execution times as its input. Since performance is typically defined as the inverse of the execution time, a smaller geometric mean translates to greater overall performance. The geometric mean is defined as:

$$\begin{aligned}\text{Geometric Mean} &= \sqrt[n]{\prod_{i=1}^{n} \text{Execution Time}} \\ &= \sqrt[n]{\prod_{i=1}^{n} \frac{\text{Instruction Count}_i \times \text{Clock Cycle}_i}{\text{Instructions Per Cycle}_i}}\end{aligned}$$

2 Order of Experimentation

3 Experiments

4 Conclusion