CSCE 692
DESIGN PRINCIPLES OF COMPUTER ARCHITECTURE
Lab 3 Report
CC1 Benchmark

Authors:
Bradley FRENCH, Micah HAYDEN, Zach MADISON, Jake MAGNESS, & Lucas MIRELES

Thursday 28th February, 2019

Abstract

This experiment sets out to evaluate design considerations and trade-offs that must be made when designing processors and their caches. This experiment considers and evaluates numerous configurations in order to determine which processor and cache configuration is optimal for given size constraints. It then compares the optimal solutions' performance per area. While it would seem that larger and more expensive hardware is always preferred, that isn't necessarily the case, especially when metrics such as cost per performance and performance per size are weighted heavily. In this experiment, optimal designs are found for processors that are limited by three different size requirements and an accompanying cache configuration for one of the processors is found.

Contents

List of Tables

1 Introduction

With the wide variety of applications and ever increasing demands that modern computers are called upon to perform, it has become increasingly important that processor/cache configurations perform optimally. Processor and cache designers must wrestle with developing high-performance processor and cache configurations while also keeping cost, size and power usage to a minimum. To reach performance benchmarks while also falling within size and cost considerations, designers must make determinations as to which features and hardware should be included in the processor. In addition, designers must then choose a cache configuration that will best compliment the proposed processor. At first glance, it would seem that more hardware always equates to a better performing processor but it has been shown that that isn't always necessarily the case. Blindly creating a processor that just barely fits within size requirements will rarely yield optimal performance and designers must make careful consideration as to what hardware and features to include in processors and caches.

In this experiment, we go through a series of processor and cache configurations to determine which design is optimal. Design parameters are isolated and then evaluated to find which ones improve the performance to cost ratio the most. These design choices are then chosen in order of performance improvement and incorporated into the designs. Each design must make trade-offs in order to meet the size requirements but through careful choice as to which features and hardware are incorporated, optimal results are achieved for each size of processor and the cache configuration.

2 Background

In the early days of computing, raw clock rate was a good measurement for processor performance. As processors became more complicated, simply using clock rate as a measurement for performance became increasingly less accurate. With processors becoming rapidly more complicated and incorporating new features, performance could no longer be accurately measured by just clock rate. The true, best measurement to evaluate processor performance is known as cycles per instruction (CPI). This value represents how many clock cycles must be performed for each instruction issued to the processor, with a lower value being better. CPI is directly related to performance because CPI and execution time are directly correlated. The only real way to determine which processor is faster is to determine which one finishes a given task first. Therefore, CPI is the measurement that is utilized in this experiment to compare processor performance. When comparing caches, the most important metrics are latency and miss rate. Latency is the time that it takes to access files that are located within the cache. A larger cache will typically have a higher latency. The other metric that is utilized when evaluating caches is miss rate. While latency is higher on larger caches, the miss rate is smaller. When making cache design considerations, it is important to find the ideal size in order to minimize miss rate and latency to improve processor performance and reduce CPI.

3 Methodology

3.1 Part A: Processor Configuration

For each simulation, the critical statistic will be the Cycles Per Instruction (CPI). Because the only "true" measure of performance is the execution time [1], this will be the best metric because we are not changing the clock cycle time or frequency. Thus, the configuration(s) with the lowest CPI, will have the best performance. We will also use the inverse of CPI as a quantifiable metric of performance for calculating a performance per unit area.

$$Performance = \frac{1}{CPI} \tag{1}$$

We will include the results of our baseline tests in this section because they informed the methodology for our optimization process; and we will show the optimization results in Section #4.1. We will begin our experimentation by running baseline tests for the different execution orders and branch predictions. We will run these tests with a machine width of 4, and with 4 of each functional unit. By maximizing the number of functional units, it will prevent any bottlenecks from under-allocating resources within the allotted amounts. These results will allow us to compare the branch prediction performance, for both in and out of order execution.

Execution Order \ Branch Prediction	Taken	Not Taken	Two Level
In Order	2.182	2.194	1.7322
Out of Order	1.6215	1.7003	1.1642

Table 1: Measured CPI of Branch predictors and execution orders

As shown in Table #1, there was no significant difference in the CPI produced by taken/not taken for either in order or out of order execution. However, for both execution orders, the 2-level predictor caused a significant increase. Also, for each branch prediction strategy, out of order execution caused significant increases compared to its in-order equivalent. This indicates that the optimal solution will most likely require out of order execution and a 2-level branch prediction strategy.

After we determined the optimal configuration with regards to execution order and branch prediction strategy, we then compared the gains provided by increasing the amount of functional units. We tested this by setting the machine width to its maximum of four to prevent a potential bottleneck, and then increased each functional unit to its maximum amount, while leaving all others at their minimums. These tests will compare to the baseline performances in Table #2, which was simulated using a machine width of four, and all functional units set to one.

Execution Order	CPI
In Order	1.8529
Out of Order	1.5061

Table 2: Baseline CPIs with 2-Level branch predictor, width = 4 and 1 of each functional unit

We will then compare the CPI improvements observed by increasing the amounts of functional units to the baseline to determine which functional units provide the most improvement/most utility. For each of the processor sizes, we will only increase the functional units producing noticeable effects. For the optimization process, we determined the following optimization priority:

1. Branch Prediction Strategy
2. Execution Order
3. Processor Width
4. Functional Unit increases (IALU has priority)

We will begin each size constraint by optimizing according to the above priority. We will then compare the result to small changes based on the Functional Unit comparisons. Finally, after determining the optimal configurations for each size, we will compare them using their performance per unit area.

3.2 Part B: Cache Optimizations

Due to the small number of possible configurations of the cache (18 total), we will run a simulation for each cache configuration. We will collect the miss rate and CPI of each cache configuration. This will guarantee the optimal solution in a feasible amount of time. We calculated the cache parameters using the following equation:

$$Cache\,Size = \#Sets \times Block\,Size \times Associativity \tag{2}$$

This allowed us to modify the simulation parameters and achieve the required cache configurations.

4 Analysis

4.1 Part A: Processor Configuration

Table #3 shows the effect of increasing the numbers of each functional unit from one to four, compared to baselines for in and out of order execution, shown in Table #2 in Section 3.1.

We ran a simulation maximizing the number of all components, with out of order execution and 2-Level branch prediction to gauge the optimal CPI, regardless of size. This produced the **optimal CPI of 1.1642** for the benchmark. However, this optimal CPI does not fall within the specified size constraints.

Execution Order \ Functional Unit	Integer ALU	Integer Multiplier	FP ALU	FP Multiplier	Memory Port
In Order	1.7323	1.8529	1.8529	1.8529	1.8529
Out of Order	1.2422	1.5061	1.5061	1.5061	1.5061

Table 3: Measured CPI of Functional Unit Increases

As shown in Table #3, the only component that produced a better result, relative to all other changes, was the Integer ALU. Therefore, we will prioritize adding the Integer ALU over all other functional units in our optimization. After determining that the Integer ALU produced the only increase relative to the other functional units, we needed to determine the effects of adding additional functional units to a system with 4 Integer ALUs. Thus, for this test, the processor had out of order execution, a 2-Level branch predictor, 4 Integer ALUs, 4 of the "Component Added", and 1 of all other functional units. These results are shown in Table #4 below.

Component Added	CPI
None	1.2422
Int. Multiplier	1.2421
Memory Port	1.1642
FP ALU	1.2422
FP Mult	1.2422

Table 4: Measured CPI of Additional Functional Units to a system with 4 Int. ALUs and a machine width of 4

As seen in Table #4, the only functional unit that produced an improvement to a 4 IALU system was the memory port. By adding the additional memory ports, the system achieved the optimal CPI of 1.1642.

The following three subsections detail the optimization of the three size constraints, following the optimization priority specified at the end of Section 3.1.

4.1.1 2600 Unit^2

We began by choosing a 2-Level branch predictor. From there, we could not afford out of order execution, so it was not considered. We then increased out machine width to 4. We could not afford any other functional units, so we tested that configuration. This produced a CPI of 1.8529.

We then just needed to verify that our selection was optimal. We tested this hypothesized optimal configuration against the configurations shown in Table #5[1]

Machine Width	Modifications	Size (Units^2)	CPI
4	None	2550	1.8529
2	None	2050	1.9585
2	2 IALUs & 2 MemPorts	2600	1.8638

Table 5: Measured CPI of Modifications - Size 2600

[1]Modifications indicates any functional unit changes from the hypothesized optimal configuration for the given size, or a switch in execution order. This definition will hold for the 4000 Unit^2 and 5000 Unit^2 tests.

4.1.2 4000 Unit^2

We began by choosing a 2-Level branch predictor, out of order execution, a machine width of two, and 4 integer ALUs. Based on our optimization criteria, after selecting 2-Level and out of order execution, we could not afford a width of 4. Thus, we selected a width of 2 and 4 integer ALUs for our hypothesized optimal solution.

Machine Width	Modifications	Size (Units^2)	CPI
2	Out of order & 4 IALUs	3910	1.4208
2	Out of order & 2 IALUs	3570	1.4208

Table 6: Measured CPI of Modifications - Size 4000

We could not afford to add a second memory port with out of order execution. Thus, the only functional unit that would provide an increase is the Integer ALU. As shown in Table #6, decreasing from 4 to 2 Int. ALUs produced no effect on the CPI. We decided to not add any other functional units because of the results outlined in Table #3: we would be increasing our size with no benefit to performance, which would reduce our performance per unit area. We also did not need to test our hypothesized optimal configuration against any in-order execution because the optimal CPI of an in-order execution would be 1.7322, as shown by Table #1.
The optimal configuration for a 4000 Unit^2 processor was a 2-Level branch predictor, out of order execution, 2 Integer ALUs, and 1 of all other functional units. This configuration produced a CPI of 1.4208, with a size of 3570 units^2.

4.1.3 5000 Unit^2

We began by selecting a 2-Level branch prediction, out of order execution, a machine width of 4, and 4 Integer ALUs. This hypothesized optimal configuration had a size of 4760.

Machine Width	Modifications	Size (Units^2)	CPI
4	Out of order & 4 IALUs	4760	1.2422
4	Out of order & 2 IALUs & 2 Memory Ports	4930	1.2186

Table 7: Measured CPI of Modifications - Size 5000

As shown in Table #7, by reducing the number of Int. ALUs to 2 to allow us to add a memory port, the performance increased from 1.2422 to 1.2186. Similarly to Section 4.1.2, we did not need to run any tests comparing our hypothesized optimal configuration to any in-order execution. Thus, the optimal configuration for a 5000 unit processor is a 2-Level branch predictor, out of order execution, a machine width of 4, 2 integer ALUs, and 2 memory ports. This configuration has a size of 4930 units^2 with a CPI of 1.2186.

4.1.4 Questions:

1. How does the performance per unit area compare among the three configurations you found? Do you think it is worth it to use the larger configurations?

   To calculate the performance per unit area, we will divide the performance as specified in Eqn. #1 by the area of each configuration[2]:

   $$Performance\,per\,unit\,area\,[PPUA] = \frac{Performance}{Area} \tag{3}$$

   (a) $PPUA_{2600\,Units} = \frac{\frac{1}{1.8529}}{2550} = 0.00021164$

   (b) $PPUA_{4000\,Units} = \frac{\frac{1}{1.4207}}{3570} = 0.00019716$

   (c) $PPUA_{5000\,Units} = \frac{\frac{1}{1.2186}}{4930} = 0.00016645$

[2] $PPUA_{X\ units}$ refers to the optimal processor under the size constraint X

While it would seem like a larger processor whose CPI is lower than the rest of the configurations would be worth it, the results above suggest that a larger configuration produces a lower performance per unit area. The $2550\,units^2$ processor has the worst performance out of the three configurations shown, by CPI. However, its performance per unit area was actually **better** than those of the two larger configurations. Thus, it is not worth it to use a larger configuration based on performance per unit area.

2. Include a table of the configurations, area, and CPI for the simulations you ran while searching for the best solutions.

   We included the table of configurations in the sections where they would be appropriate, rather than showing a single, large table here. Appendix C contains two tables of raw data from our simulations. The simulation results included in the body of the report are the simulations required to determine that our optimal solutions are optimal.

4.2 Part B: Cache Optimizations

As shown in Table #8, the miss rate consistently decreased as the block size and associativity increased. The optimal miss rate was 0.0048, which occurred with a 64KB cache, with 32 Byte blocks, and an associativity of 4. The CPI for all 16 KB caches was lower than the CPIs of all 64 KB caches. The difference in CPI is a product of the relative speed: larger memories are slower than small memories. For this simulation, the 64 KB cache had a 2 cycle access time, while the 16 KB cache had a single cycle access time. When the block size and cache size is equal, the lowest CPI and miss rate goes to the 4-way associative cache.

4.2.1 Questions:

1. What is the L1 data cache miss rate and the CPI of your benchmark for this processor configuration for each of the possible cache configurations?

Cache Size (KB)	Block Size (Bytes)	Associativity	CPI	Miss Rate
16	8	1	1.6304	0.0653
16	8	2	1.6293	0.0625
16	8	4	1.6291	0.0620
16	16	1	1.6276	0.0365
16	16	2	1.6266	0.0334
16	16	4	1.6263	0.0330
16	32	1	1.6264	0.0229
16	32	2	1.6248	0.0181
16	32	4	1.6244	0.0178
64	8	1	1.9035	0.0163
64	8	2	1.8971	0.0107
64	8	4	1.8945	0.0091
64	16	1	1.9020	0.0123
64	16	2	1.8958	0.0076
64	16	4	1.8936	0.0065
64	32	1	1.9006	0.0100
64	32	2	1.8947	0.0056
64	32	4	1.8927	0.0048

Table 8: Measured CPI and miss rate of varying cache configurations

2. Under what circumstances would a 64KB cache be more valuable than a 16KB cache for this application?

   According to the data above, each case for the 64KB cache has a higher CPI but has a lower miss rate compared to a 16KB cache. If CPI was the only aspect that mattered, a 16KB cache would always be desired. However,

the miss rate plays a large part in desired performance. There could be an application that isn't able to take advantage of spacial locality as effectively as other applications due to the nature of the task it is accomplishing. In this scenario, it would be beneficial to have a cache that is able to hold more data as there is a higher chance that data required for the next instruction will already be present in the cache. In this scenario, a larger cache could be beneficial because read and write operations to and from memory don't need to happen as often, saving time despite the longer access time.

3. Were you able to outperform the larger processors you found in Section A simply by varying the cache configuration? Do you think it would be possible to do so with more freedom to vary the cache configuration? Why or why not?

   Varying the cache configuration can provide huge performance benefits to a processor. As can be seen from 8, some cache configurations change the CPI value by roughly 0.5. A performance difference like this can easily affect the performance of a processor enough to make it better than a larger process design with a general cache configuration. Having more freedom to vary the cache configuration will allow for even greater performance benefits at the cost of further analysis and a more difficult design process (and potentially cost, power and size). Knowing the specific task that the design has to carry out can allow a processor designer to specifically tailor a cache that fits in the "sweet spot" between size, block size and associativity to provide optimal results.

4. Do you think the cache configuration would matter more or less for applications with larger data set sizes than this one?

   As the data set size increases, the cache configuration matters more. There is a design trade-off between the miss rate and the CPI. If you want to minimize the amount of misses due to a large penalty to access main memory (or a lower level of the cache), a large block size and a large cache would be preferable. With larger data set sizes, the processor has to run for longer and seemingly insignificant differences in designs can turn out to be very large performance differences. With large data sets, it is even more important to have the optimal design otherwise your processor ends up spending too much time waiting on the cache to retrieve data or the cache always has the data and the processor just isn't able to access it quick enough. The difference between waiting an extra 3 nanoseconds for one cache doesn't seem like a big deal. But, when you have to wait 3 nanoseconds for billions of instructions that you wouldn't have to had waited for in an optimal cache configuration, the time very quickly adds up.

5 Conclusion

For this experiment we set out to design a processor with the best performance for given size constraints. These configurations were then compared using their performance per unit area. In addition, we were also tasked to find the cache configuration that would perform the best in conjunction to our processor design with an area less than or equal to 2600 Units^2. Through experimentation and isolation of variables, we were able to determine which processor design considerations that made the largest impact on processor performance. Early experimentation showed that two-level branch prediction and out of order execution yielded the greatest performance benefit relative to their cost in area. It was then determined that prioritizing increasing processor width over increasing the amount of functional units yielded the best results. Finally, we found that the only functional units that improved processor performance were integer ALUs and memory ports. It was discovered that there was a sweet spot between having enough integer ALUs to perform operations and enough memory ports to perform read and write operations quick enough to keep up with the processor. An example of this can be found on the optimal configuration for a processor under 5000 Units^2 where two integer ALUs were removed in favor of adding another memory port. By isolating variables and finding the most imperative design considerations to improve performance, we found the following cache configurations to be optimal for the 2600, 4000 and 5000 Units^2 listed in the table below. Note that every design utilized 2 level branch prediction.

Processor Size (Units^2)	Machine Width	Modifications	CPI
2600	4	None	1.8529
4000	4	Out of Order & 2 IALUs	1.4208
5000	4	Out of Order & 2 IALUs & 2 Memory Ports	1.2186

Table 9: Optimal Processor Designs

Cache optimization proved to a bit easier than processor optimization as the performance changes from design changes are easier to predict. Due to the reduced number of potential configurations, we also had the benefit of being able to brute-force optimize the cache configuration for the processor. Typically, cache configurations that are smaller in terms of storage tend to have lower access rates but higher miss rates. Larger caches tend to have slower access times but have lower miss rates. The data we collected is consistent with the aforementioned generalities as shown in 8. With CPI values collected from the experiment, it is simply a matter of picking the cache configuration with the lowest CPI. The cache configuration that performed the best was a 16KB cache with 32 Byte blocks and an associativity of 4. This configuration allows for a smaller cache to reduce access time while also minimizing misses with higher associativity and reducing the number of reads and writes with a large block size.

In this experiment we have found the optimal processor design for each of the respective sizes: 2600 Units^2, 4000 Units^2 and 5000 Units^2. In addition, we found the cache configuration that performs best with the processor design that is less than or equal to 2600 Units^2. It is important to remember, however, that these optimal design only hold true for the specific benchmark we ran these designs against. In a different benchmark where there are many floating point multiply operations for example, these same designs would likely perform very poorly. As an engineer, it is important to always remember to tailor your design for the specific role it is designed to fill.

References

[1] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Sixth Edition: A Quantitative Approach*, 6th ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2017.

Appendix A: Lessons Learned

Bradley French

1. Creating a bigger chip doesn't necessarily mean better results; in some cases, it even means worse. While this depends on your configurations of the various elements of the chip and the software ran against it, there were configurations in this project where the larger chips were outperformed by the smaller chips.

2. The trade-off between CPI and miss rate can be significant on the performance of a cache. Varying between larger caches for a lower miss rate or smaller caches for a smaller CPI can be a difficult concept to understand initially.

3. While concepts of the set-associative caches were already understood, altering each part of the cache can truly alter the performance of the cache. This includes cache size, block size, set associativity, number of sets, and the list continues. Altering some of these, such as the block size, without altering the cache size, can truly affect the outcome of the CPI and miss rate in a positive direction.

Micah Hayden

The main lesson I learned was the importance of isolating parameters. The amount of data the simulator produces, combined with the possible permutations of the simulation produced a significant amount of data. By carefully looking for ways to isolate variables and compare results, it very quickly reduces the size of the search space for the optimal solution. Also, book keeping is very important. As we increased the number of simulations, we needed to keep track of all of the simulation results in a way in which they were easily accessible.

Zach Madison

1. Larger caches can actually be detrimental and lead to a lower CPI because there are more blocks to search through in order to determine if there is a hit or a miss.

2. Although the configuration in our results with the lowest CPI was one of the largest sizes, bigger is not always better. Depending on the configurations of the smaller size chips, they could and did outperform those of the larger size chips. One example of this would be increasing the number of integer ALUs without increasing the width

3. By simply changing logical aspects of the cache structure such as blocksize and associativity, performance can be drastically improved to surpass caches that have larger sizes.

Jake Magness

This lab made me realize that it is extremely important to understand the problem that your design is meant to solve. It was pretty quickly realized which branch prediction technique/processor width/execution order performed the best but the functional units were a bit more difficult to isolate. Once we realized that the benchmark heavily favored integer ALUs, we were then able to tweak and find the sweet spot between having enough ALUs to perform operations and enough memory ports to keep up with the processor.

Lucas Mireles

I learned the importance of planning before jumping into the execution of tests. Specifically, writing out and thinking through the methodology helped plan the course of simulations required. If we had thought out our steps more carefully before starting, it would have simplified the task at hand.

Appendix B: Simulator Script

The below script was used for running the processor width simulations.

```
#!/bin/bash

date
echo "starting cc1 in-order, taken width 1, intalu 1, intmult 1, memprt 1"
sim-outorder -fetch:ifqsize 1 -decode:width 1 -issue:width 1 -commit:width 1 -res:ialu 1 \
-res:imult 1 -res:memport 1 -res:fpalu 1 -res:fpmult 1 -issue:inorder true -bpred taken \
-redir:sim "Baseline_Taken_Inorder.txt" cc1.ss -O 1stmt.i
echo "done"

diff 1stmt.s 1stmt.s.ref
echo "done with diff check"
date

date
echo "starting cc1 in-order, not taken, width 1, intalu 1, intmult 1, memprt 1"
sim-outorder -fetch:ifqsize 1 -decode:width 1 -issue:width 1 -commit:width 1 -res:ialu 1 \
-res:imult 1 -res:memport 1 -res:fpalu 1 -res:fpmult 1 -issue:inorder true -bpred nottaken \
-redir:sim "Baseline_NotTaken_InOrder.txt" cc1.ss -O 1stmt.i
echo "done"

diff 1stmt.s 1stmt.s.ref
echo "done with diff check"
date

date
echo "starting cc1 in-order, not taken, width 1, intalu 1, intmult 1, memprt 1"
sim-outorder -fetch:ifqsize 1 -decode:width 1 -issue:width 1 -commit:width 1 -res:ialu 1 \
-res:imult 1 -res:memport 1 -res:fpalu 1 -res:fpmult 1 -issue:inorder true -bpred "2lev" \
-redir:sim "Baseline_2Lev_InOrder.txt" cc1.ss -O 1stmt.i
echo "done"

diff 1stmt.s 1stmt.s.ref
echo "done with diff check"
date
```

The below script was used for modifying the cache parameters:

```
#!/bin/bash

date
echo "starting cc1 "
sim-outorder -fetch:ifqsize 4 -decode:width 4 -issue:width 4 -commit:width 4 -res:ialu 1 \
-res:imult 1 -res:memport 1 -res:fpalu 1 -res:fpmult 1 -issue:inorder true -cache:dl1:2048:8:1:l \
-bpred "2Lev" -redir:sim "16KB_Block8_Assoc1.txt" cc1.ss -O 1stmt.i
echo "done"
```

Appendix C: Raw Data Tables

Table #10 was a test to determine which branch prediction strategy was the most effective. We decided to re-run the execution order/branch prediction tests while maximizing the processor's width and # of functional units to prevent any adverse effects from having a limited system.

Execution Order	Branch Prediction	CPI	Size
In Order	Taken	2.5224	1350
In Order	Not Taken	2.5224	1350
In Order	2-Level	2.0755	1800
Out of Order	Taken	2.3956	2295
Out of Order	Not Taken	2.3956	2295
Out of Order	2-Level	1.976	3060

Table 10: In Order Branch Prediction Results

This result echoed the data analyzed from the full-system test shown in Table #1, indicating that 2-Level was the most important factor to select.
The following table shows the miscellaneous simulations which were utilized prior to formalizing the methodology[3].

Execution Order	Branch Prediction	Machine Width	Functional Units	CPI	Size
In Order	2-Level	2	2 Int. ALUs, 2 Mem. Ports, & 2 FP ALUs	1.8638	2600
In Order	2-Level	2	No Increase	1.9585	2050
In Order	Taken	4	No Increase	1.85429	2550
In Order	Taken	2	2 of all Func. Units	2.3054	2600
Out of Order	2-Level	4	No Increase	1.5061	4250
Out of Order	2-Level	2	No Increase	1.5746	3400
In Order	2-Level	2	No Increase	1.9585	2050
Out of Order	2-Level	2	2 Int. ALUs, 2 FP ALUs	1.4207	3825

Table 11: Miscellaneous Results

[3]The Functional Units column indicates any functional units increased from the minimum of 1.