# CMPEN 431

# Project 1

Kyle Ostrowski and Sarah Tickner

# Provided Framework

The provided framework and its components enable design space exploration by easily allowing us to change cache sizes and subsequent latencies. Although adjusting every parameter and running tests on it is not possible because of timing constraints, the heuristic we used allowed us to look at different parameters to find a good design by testing them in a consistent way.

# Chosen Design Point

Our Penn State IDs are 907056774 and 931701543 which added together and mod by 24 is 21 so the order in which we adjusted the different configurations was Floating Point Unit, Cache, Core, and finally Branch Predictor. Using this order of exploration, the design space chosen by our DSE for performance was configuration 0 0 2 2 0 6 0 2 3 1 0 0 4 2 3 1 5 4 with the best EDP=3.91484e-08 and the best time=0.000194057. The design space chosen by our DSE for energy was configuration 0 0 2 2 0 5 0 1 3 1 0 0 3 3 2 1 4 3 with the best EDP=3.84011e-08. and the best Time=0.000197122. The chosen configurations are shown in the table.

# Plots

Plot A took the data output by the execution time and energy efficiency logs and plotted the Normalized Geometric Mean Execution Time (NGET) for each design space that was explored. The execution time data is shown in orange while the energy efficiency data is shown in blue. The energy efficiency consistently shows lower execution time however, the overall pattern of the data is similar for both.

Plot B shows the normalized geometric mean of the Energy-Delay product for each design space that was explored. Again this graph pulled data from both the execution time and energy efficiency logs. Again the energy efficiency data has a consistently lower EDP.

Plot C shows the normalized execution time compared to the benchmark for the final chosen design to optimize the performance.

Plot D shows the normalized EDP to the benchmark for the final design to optimize EDP.

# Table

|  |  |  |
| --- | --- | --- |
| Parameter | Performance | EDP |
| Width | Value = 0  Why = This corresponds to 1, or 8 bytes. A larger width corresponds to longer cycle times and it is optimized at the lowest value. | Value = 0  Why = This corresponds to 1, or 8 bytes. A larger width corresponds to more power being used so it is optimized at the lowest value. |
| Scheduling | Value = 0  Why = This means that it is in order so instructions received first are acted on first | Value = 0  Why = This means that it is in order and thus it uses less power. |
| L1block | Value = 2  Why = This corresponds to 32 bytes. It must be at least the size of the width. | Value = 2  Why = This corresponds to 32 bytes. It must be at least the size of the width. |
| Dl1sets | Value = 2  Why = This means there are 128 sets in L1 Data. | Value = 2  Why = This means there are 128 sets in L1 Data. |
| Dl1assoc | Value = 0  Why = This means the associativity is 1. | Value = 0  Why = This means the associativity is 1. |
| Il1sets | Value = 6  Why = This means there are 2048 sets in L1 Instruction. | Value = 5  Why = This means there are 1024 sets in L1 Instruction. |
| Il1assoc | Value = 0  Why = This means the associativity of L1 is 1. | Value = 0  Why = This means the associativity of L1 is 1. |
| Ul2sets | Value = 2  Why = The number of sets in Unified L2 is 1024. | Value = l  Why = The number of sets in Unified L2 is 512. |
| Ul2block | Value = 3  Why = The number of bytes in Unified L2 is 128. | Value = 3  Why = The number of byes in Unified L2 is 128. |
| Ul2assoc | Value = 1  Why = The associativity of Unified L2 is 2. | Value = 1  Why = The associativity of Unified L2 is 2. |
| Replacepolicy | Value = 0  Why = This corresponds to the Least Recently Used replacement policy which makes more sense than FIFO or random because it is likely that if the memory is accessed it is likely to be accessed again | Value = 0  Why = This corresponds to the Least Recently Used replacement policy which makes more sense than FIFO or random because it is likely that if the memory is accessed it is likely to be accessed again |
| Fpwidth | Value = 0  Why = This is the first value that we are exploring and the larger this is the longer the cycle time is so it is optimized at the lowest value. | Value = 0  Why = This is the first value that we are exploring and the larger this is the more pipeline leakage there is so it is optimized at the lowest value. |
| branchsettings | Value = 4  Why = bpred comb -bpred:comb 1024 | Value = 3  Why = bpred 2lev -bpred:2lev 4 256 8 0 |
| Ras | Value = 2  Why = The return address stack has a size of 4 entries. | Value = 3  Why = The return address stack has a size of |
| Btb | Value = 3  Why = The branch target buffer is 514 sets with an associativity of 4. | Value = 2  Why = The branch target buffer is 256 sets with an associativity of 8. |
| Dl1lat | Value = 1  Why = The L1 D latency is 2 because the Dl size is 4. | Value = 1  Why = The L1 D latency is 2 because the Dl size is 4. |
| Il1lat | Value = 5  Why = This value is entirely dependent on the Il1block size, which is 32 bytes so the latency is 5. | Value = 4  Why = |
| Ul2lat | Value = 4  Why = The Ul2 latency is 9 because the Ul12 size is 512. | Value = 3  Why = The Ul2 latency is 8 because the Ul12 size is 512. |

# More Sophisticated Heuristic

If we continued with our heuristic but instead of ‘freezing’ values after we found the optimal one, we tested every single combination, we would find the best design. However, this is a difficult problem because of the constraint on the number of design spaces we can explore (1000). It is necessary because otherwise, exploring them could take a very long time, up to tens of years to try every single combination of the parameters listed here. In that time, technology could improve and the results would then be meaningless. Not to mention, the inefficiency of that process. [3]

A better heuristic to more efficiently explore the design space might be to look at the adjustments that made the largest impact in performance and EDP for our heuristic and focus more on those, and less on the other parameters. For example, the branch target buffer had a significant impact, so it may be worth exploring that more, and the width a little bit less.

Additionally, this is a problem that has been researched before, JavasheelGowda wrote her thesis about Design Space Exploration, looking specifically at high level synthesis or HLS. Which is a tool that can considerably speed up the process in an efficient way. [1][2]

# Insights

Some insights we gained while working on this project is how long it can take to run these tests. When designing hardware, these are important considerations to creating a reliable and consistent design that uses resources in an efficient and timely manner so it really important to use intelligent algorithms to explore the necessary parameters.

# Resources

[1] L. Ferretti, G. Ansaloni and L. Pozzi, "Cluster-Based Heuristic for High Level Synthesis Design Space Exploration," in IEEE Transactions on Emerging Topics in Computing, vol. 9, no. 1, pp. 35-43, 1 Jan.-March 2021, doi: 10.1109/TETC.2018.2794068.

[2] JayasheelGowda, Monica. “Design Space Exploration Using Heuristic Algorithms.” *Eugene McDermott Library*, The University of Texas at Dallas , Aug. 2018, utd-ir.tdl.org/handle/10735.1/6256.

[3] Panerati J., Sciuto D., Beltrame G. (2017) Optimization Strategies in Design Space Exploration. In: Ha S., Teich J. (eds) Handbook of Hardware/Software Codesign. Springer, Dordrecht. https://doi.org/10.1007/978-94-017-7267-9\_7

# Closing Comments

The code for our design space exploration was correct, but due to some instability in the simulation software, it yielded some odd results.