Ryan Talalai

CMPEN 341

Professor Kiwan Maeng

16 December 2022

Design Space Exploration

Introduction

The goal of this assignment is to explore and evaluate a design space of computer architecture to find a design that is optimized for performance and energy efficiency. The SimpleSclar simulator and provided framework code allows us to simulate a DSE. There are four different areas to explore: BPU, Cache, FPU, and Core. In total, there are 18 configurations that can be changed, 3 of which are dependent. Our code allows us to change one of these configurations at a time, and then simulate execution times and energy consumption to see if the change made our design more optimized. When the optimal configuration has been set, it will move to the next configuration and repeat this loop process. This allows us to come up with an optimized configuration set without having to test every possible combination, as that would take way too much time. After simulating, the data can be analyzed to see how each benchmark improves optimization, and we can compare it to the default configuration to quantify the improvement.

Note: All the code implementations and data log files are submitted in "final_code_ryantalalai.zip"

Chosen Design Point for Best Execution Time:

The index of this design point is: 0 0 2 2 0 6 0 2 3 1 0 0 4 3 1 1 5 4

Which corresponds to { width=1, scheduling=inorder, l1block=32, dl1sets=128, dl1assoc=1, il1sets=2048, il1assoc=1, ul2sets=1024, ul2block=128, ul2assoc=2, replacepolicy=LRU, fpwidth=1, branchsettings= -bpred comb -bpred:comb 1024= , ras=8, btb=256 8, dl1lat=2, il1lat=6, ul2lat=9 }

Chosen Design Point for Best Energy Efficiency:

The index of this design point is: 0 0 2 2 0 5 0 1 3 1 0 0 4 3 1 1 4 3

Which corresponds to { width=1, scheduling=inorder, l1block=32, dl1sets=128, dl1assoc=1, il1sets=1024, il1assoc=1, ul2sets=512, ul2block=128, ul2assoc=2, replacepolicy=LRU, fpwidth=1, branchsettings= -bpred comb -bpred:comb 1024, ras=8, btb=256 8, dl1lat=2, il1lat=5, ul2lat=8 }

See table below for more information:

Data Analysis

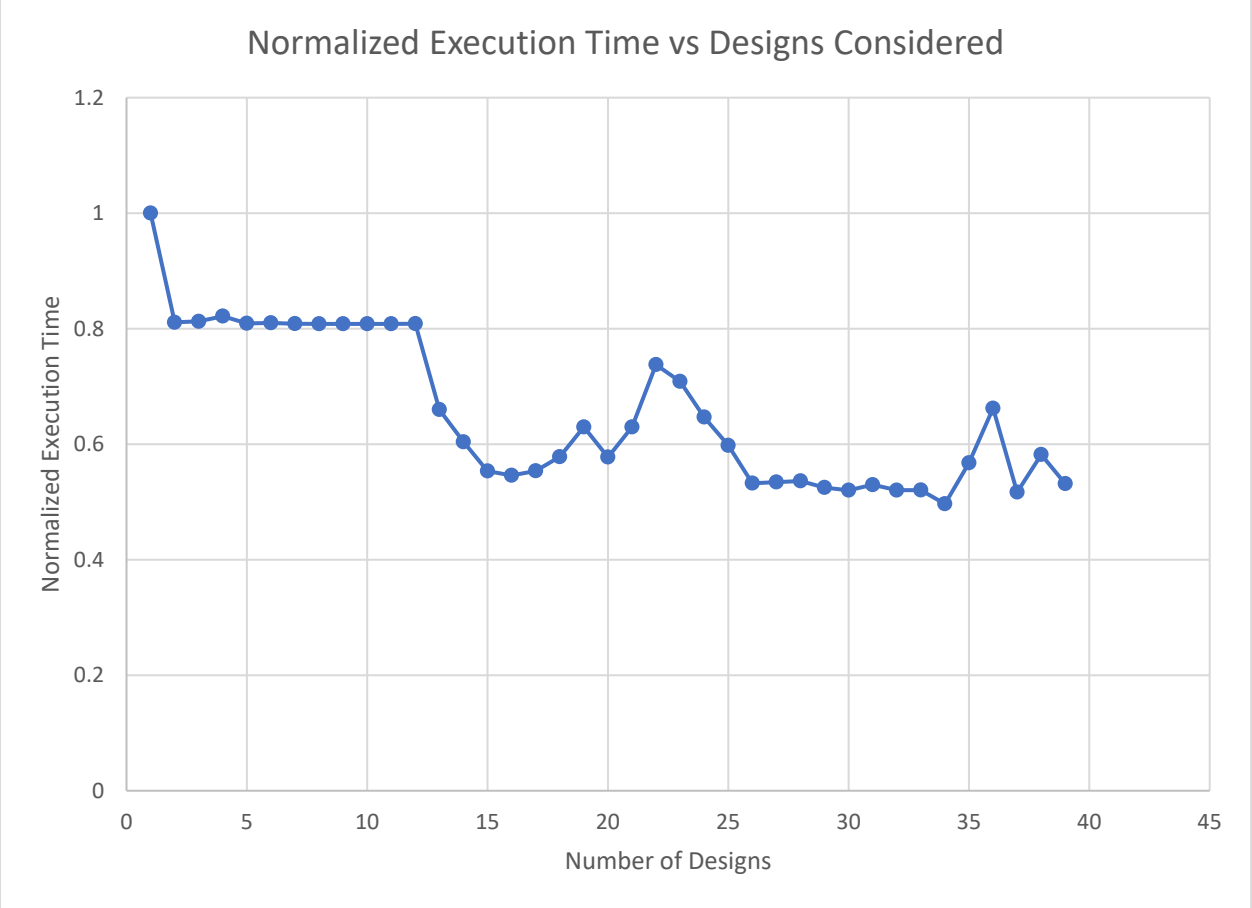
Parameter	Performance	EDP
width	Value = 1 Smaller instruction size can reduce errors and hazards.	Value = 1 Less instructions per cycle can decrease computing power.
scheduling	Value = -issue:inorder true -issue:wrongpath false In order scheduling can give better performance if rescheduling takes long.	Value = -issue:inorder true -issue:wrongpath false Reordering instructions uses more computing power.
l1block	Value = 32 Smaller caches are faster and easier to access.	Value = 32 Smaller block sizes use less computing power.
dl1sets	Value = 128 Small caches can be accessed quicker.	Value = 128 Smaller number of sets uses less computing resources.
dl1assoc	Value = 1 Easier and faster to match cache pairings.	Value = 1 Multi-level assoc use more computing resources.
il1sets	Value = 2048	Value = 1024

	Increase performance by offering a larger area to store.	Smaller number of sets uses less computing resources.
il1assoc	Value = 1 Lower miss rate increases performance.	Value = 1 Multi-level assoc use more computing resources.
ul2sets	Value = 1024 More storage means cache has more data to produce a cache hit	Value = 512 Smaller number of sets uses less computing resources.
ul2block	Value = 128 Faster accesses.	Value = 128 Smaller block sizes use less computing power.
ul2assoc	Value = 2 Lower miss rate increases performance.	Value = 2 Multi-level assoc use more computing resources.
replacepolicy	Value = LRU Increases performance because there are faster accesses to the most used locations.	Value = LRU When correctly hits, fewer computing resources are utilized.
fpwidth	Value = 1	Value = 1

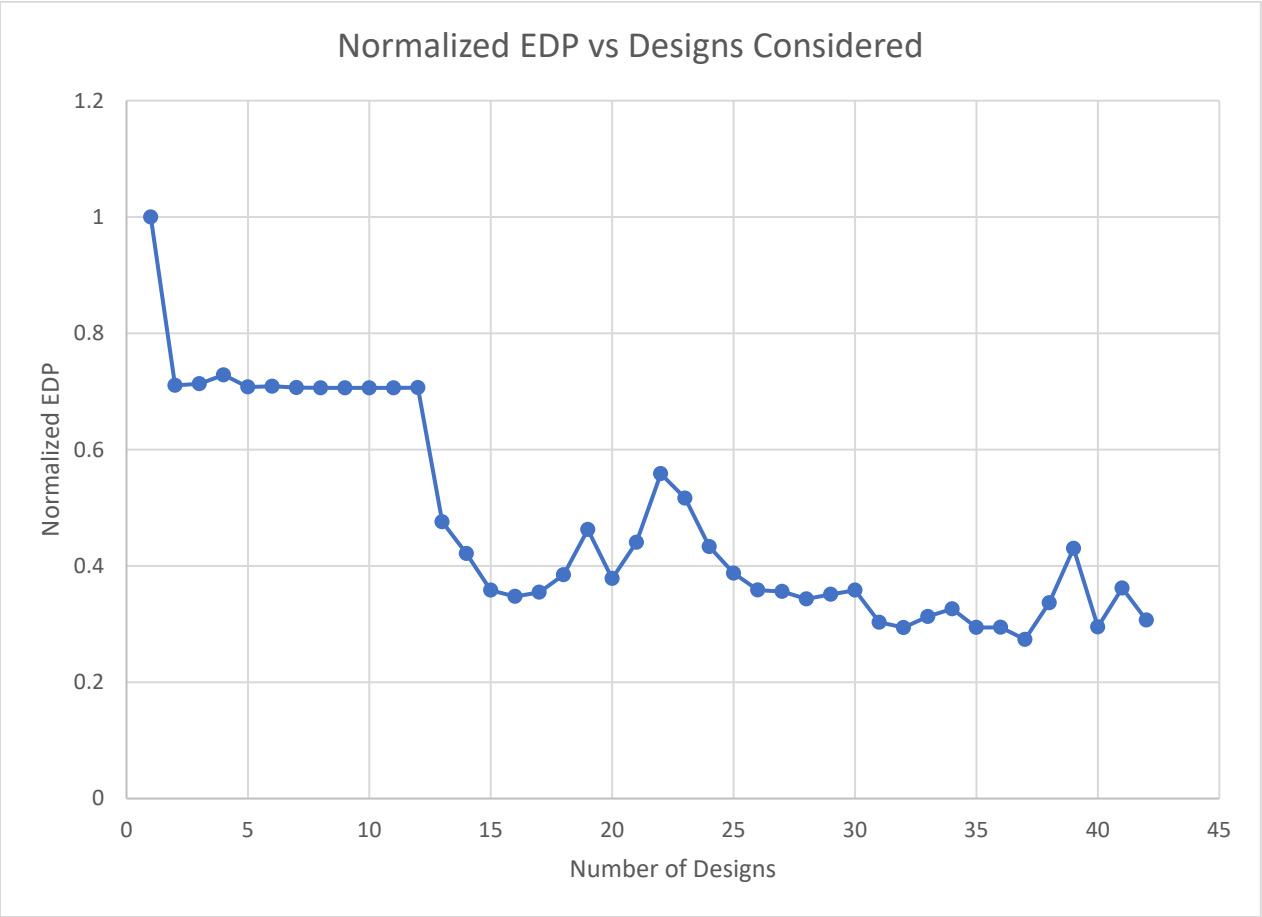
	Minimizes the number of unused bits to increase performance.	Uses proper width to increase efficiency.
branchsettings	Value = -bpred comb -bpred:comb 1024 Offers a higher correct prediction rate and increases performance.	Value = -bpred comb -bpred:comb 1024 Predictor combination is more efficient than single predictor
Ras	Value = 8 Return more address bytes at once can increase performance.	Value = 8 Returning more address bytes can increase efficiency.
Btb	Value = 256, 8 Lower miss rate and increases performance.	Value = 256, 8 Less misses means increased efficiency
dl1lat	Value = 2 Dependent on dl1 size	Value = 2 Dependent on dl1 size
il1lat	Value = 6 Dependent on il1 size	Value = 5 Dependent on il1 size
ul2lat	Value = 9 Dependent on ul2 size	Value = 8 Dependent on ul2 size

Plots

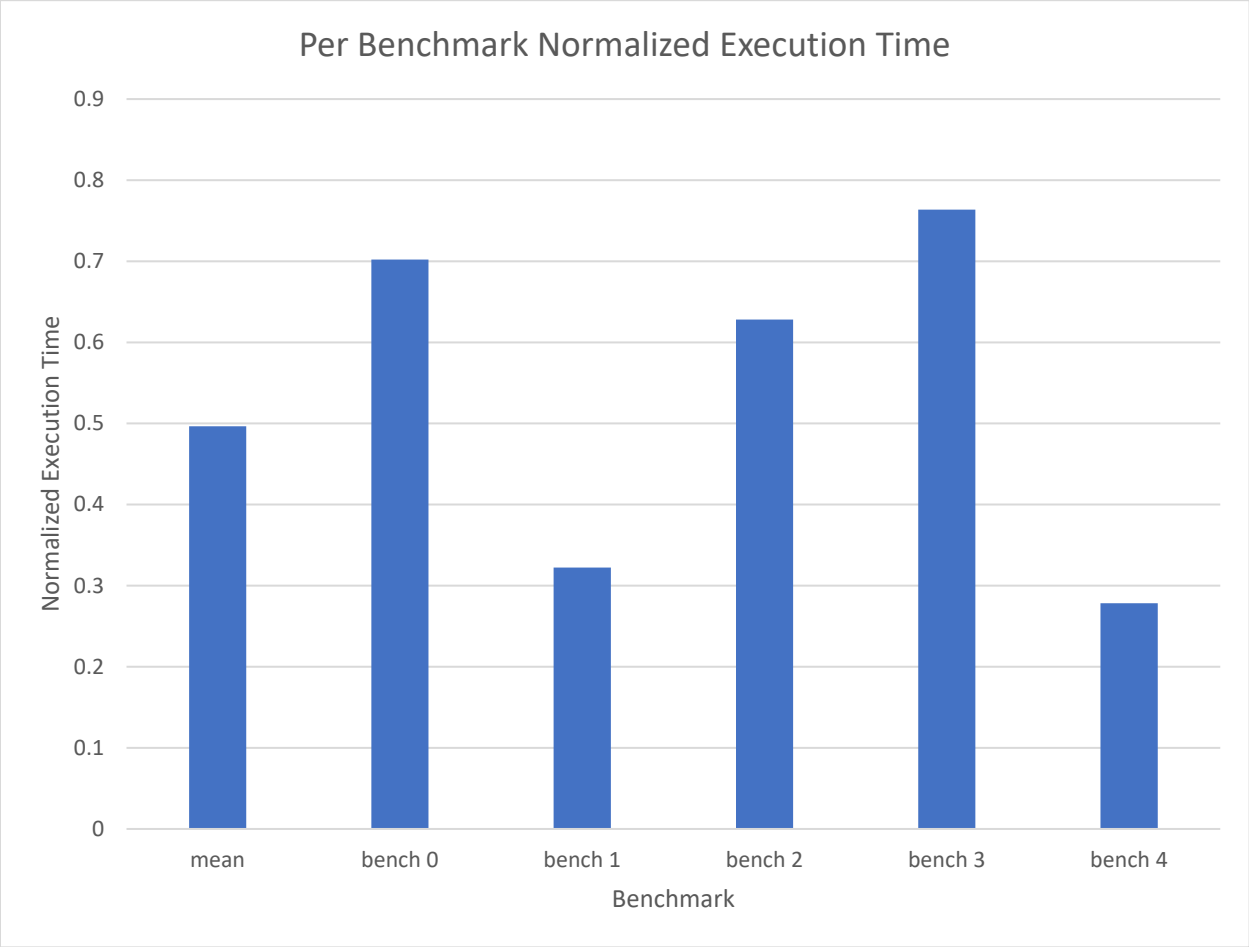
Line plot of normalized geomean execution time (y axis) for each considered de- sign point vs. number of designs considered (x axis)



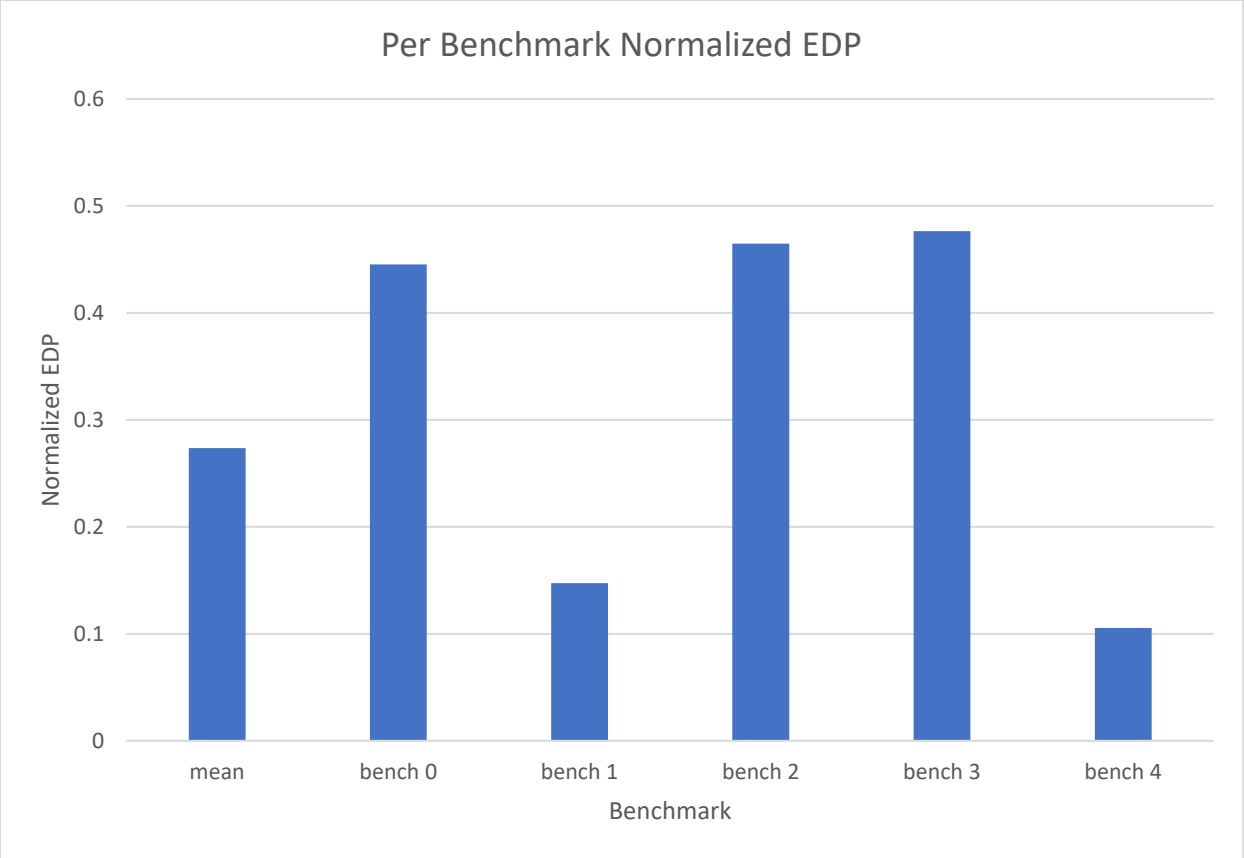
Line plot of normalized geomean of energy-delay product (y axis) vs number of designs considered



Bar chart showing normalized per-benchmark execution time and geomean normalized execution time for the best performing design



Bar chart showing per-benchmark normalized energy-delay product and geomean normalized energy delay product for the most energy-efficient design found



A More Sophisticated Heuristic:

The current heuristic algorithm consists of changing one configuration at a time, starting with the branch predictor settings. If changing it increases performance or efficiency, then we set that configuration to that setting. After going through every configuration one by one, an optimized design space is found in less than 1000 iterations.

To make this current design more sophisticated, I think it would be better to start with cache settings. Since cache sizes, hit/miss rates, and associativity can have huge performance implications, starting with these would be more optimal, and a better performing design space could be found by prioritizing these configurations. However, we are still using the same algorithm, so chances are the performance will be very close anyways.

Creating a more sophisticated heuristic with a new algorithm would be a way to achieve a noticeable performance increase. One potential algorithm design could be changing every configuration in the same iteration, permitting it is valid. It can then look at the execution time to see if it made an improvement. If it did, we can advance. If not, then it can look back at the design space that had higher performance, and change one less configuration, to see if the config changed resulted in the decrease in performance. All in all, this algorithm might take a lot of cycles to give an optimized result, but I think it is worth testing in the simulator.

Gained Insights:

Throughout this project, I gained a lot of knowledge about computer architecture. Before, I was not sure what a design space exploration was, or why it is useful. Now I know it is a process for finding a solution that is very close to the best solution. This is very useful because a very good solution is found without having to do millions of simulations.

Another insight I gained was working with Linux machines remotely from my Windows 10 laptop. Using VPNs and connecting to Linux Machines through a terminal is something useful I can use in my future computer engineering path. Compiling code with make files and testing it in the terminal was also something I found useful.

Perhaps the biggest insight I gained was learning how all the parts of the framework code worked together to enable a DSE. I never used shell scripts before so reading through the "runprojectsuite.sh" file was something new for me.

Lastly, interpreting and analyzing the log data was interesting to me. I found it insightful to see how different configurations would affect execution times and EDP.

Additional Resources:

https://en.wikipedia.org/wiki/Design_space_exploration

https://courses.cs.washington.edu/courses/cse471/06sp/hw/simpleScalar3.0guide.pdf

https://www.youtube.com/watch?v=iOGqHXfL7hU&ab_channel=SethuJose

Final Comments:

Upon finishing this project, I feel my knowledge about computer architecture has grown and I gained valuable and applicable knowledge that I will use in my future career. Thank you for all your insights.