CMPEN431
Project2 Report
Name: Wei Cheng

1. In this project, we use SimpleScalar as an evaluation engine to explore the design space and find the best design for both execution time and EDP. Our design space is constrained by the provided shell script wrapper with 18 dimensional configurations and we use a 5-benchmark suit as the workload. Our simple heuristic assumes that there is no relation between two dimensions. So, we follow a baseline configuration and evaluate all possible design point by changing only one dimension at a time (fix all other dimensions). In this way, we can result in a "good enough" design configuration.
2. Best Performance:

   After running "./DSE performance", I got the configuration for best performance:

   "0 1 0 3 0 8 0 3 2 2 0 4 4 1 2 0 5 6"; The following table shows the corresponding actual value for each dimension in the configuration.

width	scheduling	l1block	dl1sets	dl1assoc	il1sets	il1assoc	ul2sets	ul2block
1	Inorder false Wrongpath true	8	256	1	8192	1	2048	64
ul2assoc	replacepolicy	tlbsets	branchsettings	ras	btb	dl1lat	il1lat	ul2lat
4	LRU	64	-bpred comb -bpred:comb 1024	2	512 4	1	6	11

   Best EDP:

   After running "./DSE energy", I got the configuration for best EDP:

“0 1 0 3 0 7 0 2 2 2 0 1 4 2 3 0 4 5”; The following table shows the corresponding actual value for each dimension in the configuration.

width	scheduling	l1block	dl1sets	dl1assoc	il1sets	il1assoc	ul2sets	ul2block
1	Inorder false Wrongpath true	8	256	1	4096	1	1024	64
ul2assoc	replacepolicy	tlbsets	branchsettings	ras	btb	dl1lat	il1lat	ul2lat
4	LRU	8	-bpred comb -bpred:comb 1024	4	1024 2	1	5	10

3. Table

Parameter	Performance	EDP
width	value = 1 why = small fetch width led to small clock cycle time. Since execute time = cycle time * CPI * instruction count, small fetch width can increase performance.	value = 1 why = small fetch width led to low core leakage power and low energy per committed instruction. So, it can reduce energy.
scheduling	value = Inorder false Wrongpath true why = Out of order executes every instruction it can as quickly as possible without waiting for previous instruction to finish unless there are some data dependencies. So, it has better performance than in-order.	value = Inorder false Wrongpath true why = wrongpath instruction prefetching can help reduce energy.

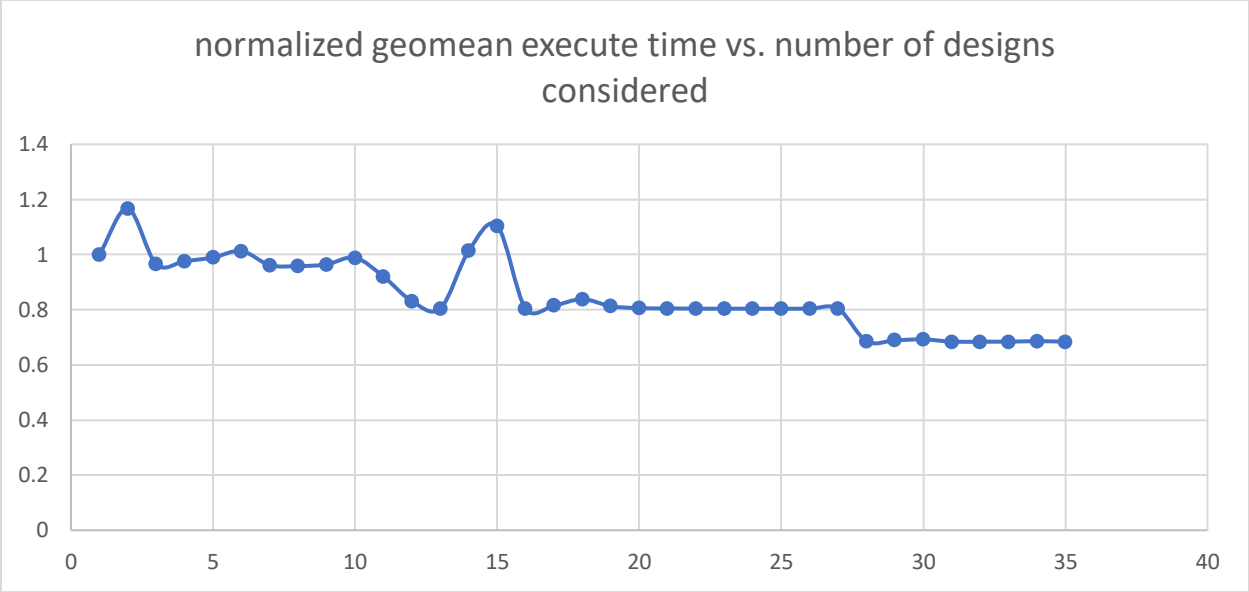
l1block	value = 8 why = small block size takes less time to access	value = 8 why = small block size has low access energy and low leakage
dl1sets	value = 256 why = small number of sets has low access time.	value = 256 why = small number of sets has low access energy and low leakage.
dl1assoc	value = 1 why = small associativity has low access time.	value = 1 why = small associativity has low access energy and low leakage.
il1sets	value = 8192 why = large cache sets can avoid capacity misses, because it can increase cache size. So, it can increase performance.	value = 4096 why = large cache sets can avoid capacity misses, because it can increase cache size. So, it can reduce energy.
il1assoc	value = 1 why = small associativity has low access time.	value = 1 why = small associativity has low access energy and low leakage.
ul2sets	value = 2048 why = small number of sets has low access time.	value = 1024 why = small number of sets has low access energy and low leakage.
ul2block	value = 64 why = large block size can avoid compulsory misses. So, it can increase performance.	value = 64 why = large block size can avoid compulsory misses. So, it can reduce energy.

ul2assoc	value = 4 why = large associativity can reduce conflict misses. So, it can increase performance.	value = 4 why = large associativity can reduce conflict misses. So, it can reduce energy.
replacepolicy	value = LRU why = programs tend to visit the same address multiple times in a short period.	value = LRU why = programs tend to visit the same address multiple times in a short period.
tlbsets	value = 64 why = large tlbsets can avoid capacity misses. So, it can increase performance.	value = 8 why = small tlbsets has low access energy and low leakage.
branchsettings	value = -bpred comb -bpred:comb 1024 why = comb has high branch prediction accuracy	value = -bpred comb -bpred:comb 1024 why = comb has high branch prediction accuracy
ras	value = 2 why = small size of ras means there are few function calls in the progrom.	value = 4 why = large size of ras can store more return addresses. So, it can support more function calls.
btb	value = 512 4 why = large associativity can avoid conflict misses. So, it can increase performance.	value = 1024 2 why = large btbsets can avoid capacity misses. So, it can reduce energy.
dl1lat	value = 1	value = 1

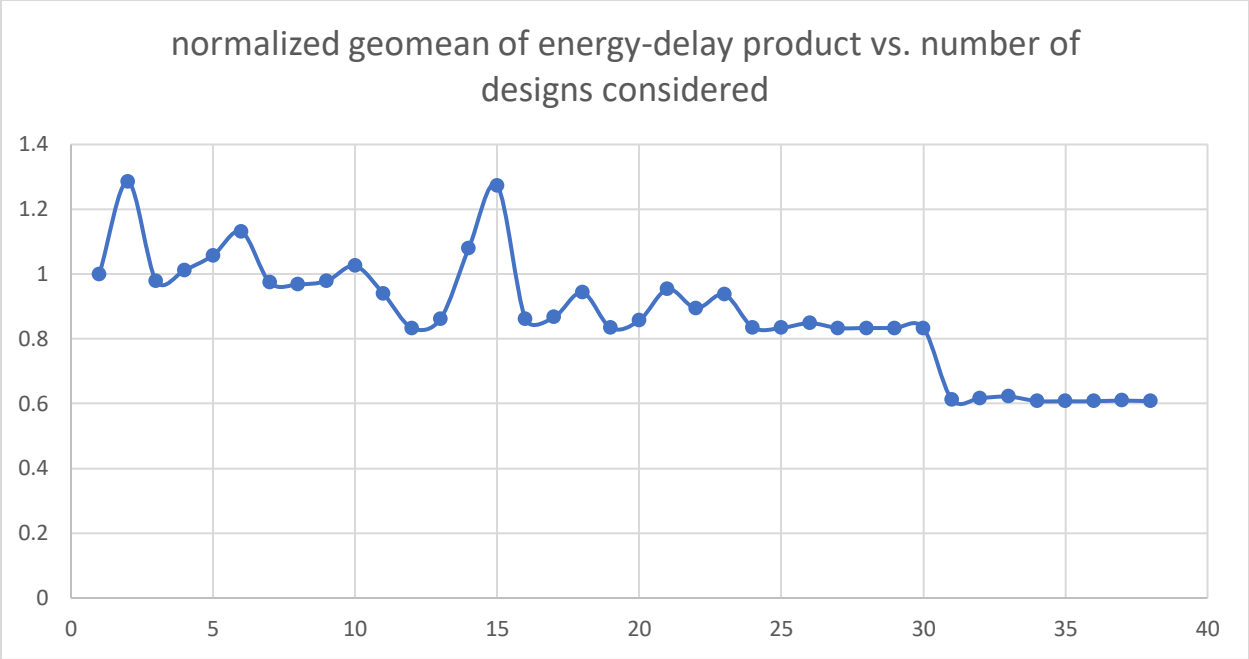
	why = calculated by L1 block, dl1sets and dl1assoc.	why = calculated by L1 block, dl1sets and dl1assoc.
il1lat	value = 6 why = calculated by L1 block, il1sets and il1assoc.	value = 5 why = calculated by L1 block, il1sets and il1assoc.
ul2lat	value = 11 why = calculated by L2 block, ul2sets, ul2assoc.	value = 10 why = calculated by L2 block, ul2sets, ul2assoc.

4.

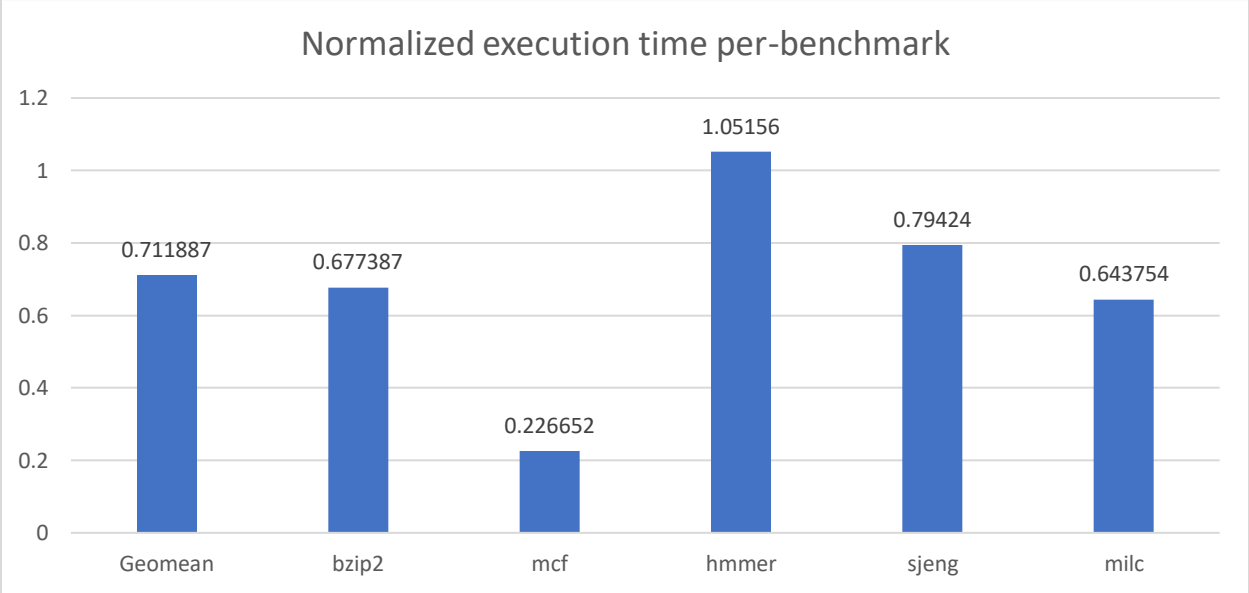
A. Line plot of normalized geomean execution time for each considered design point vs. number of designs considered



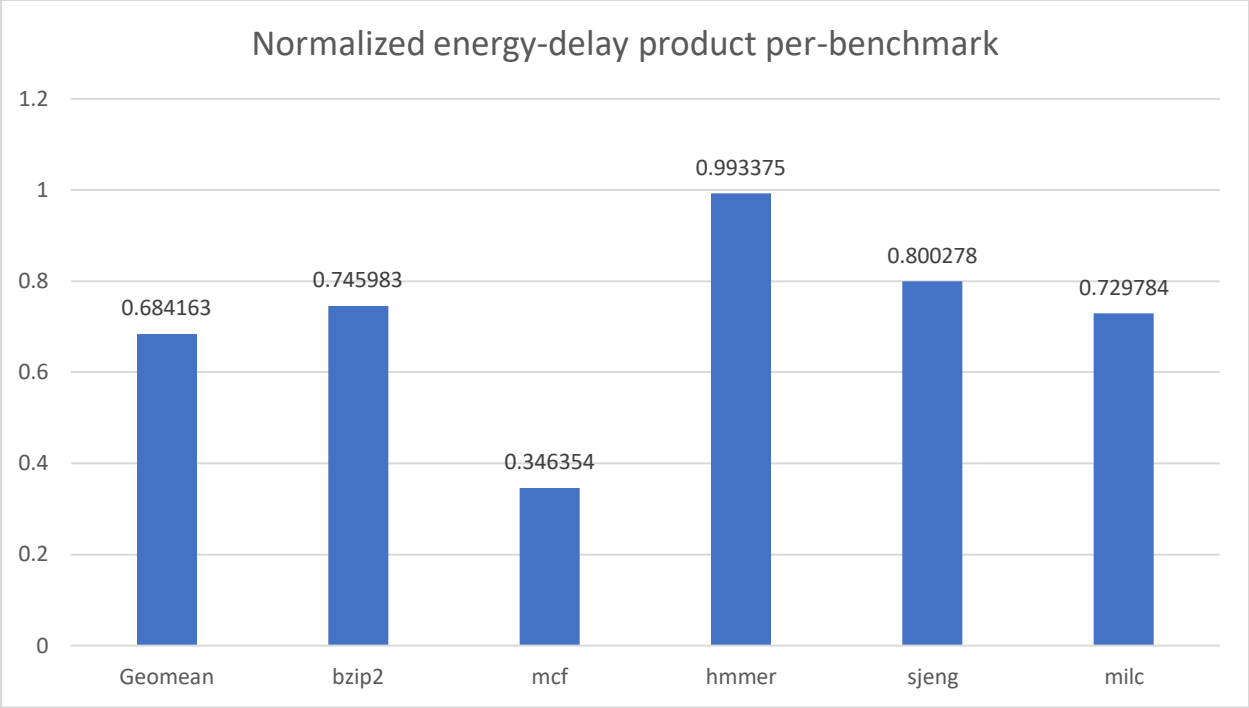
B. Line plot of normalized geomean of energy-delay product vs. number of designs considered



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



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



5. The limitation of design space exploration is 1000 design points; however, our heuristic only iterates less than 100 times. There are a lot of design points we can explore before we reach the limitation. Therefore, I think a better heuristic is that instead of changing only one dimension at a time, we can try to change two dimensions at the same time with all other dimension fixed. In this way, we can explore more design point and more likely to find a better design point. In addition to this, I think another way we can do is to keep calling our simple heuristic function and let the output in the last call be the baseline for the next call. So that, we can keep improving the current-got best design points. Since, originally, our choice of baseline may not be reasonable, in this way we can get better and better baseline.
6. I learned a lot of things in this project. Firstly, I learned how to perform a design space exploration. If we want to perform an exhaustive exploration and find the absolute best

design point, we have to try all the possible combinations of the parameter values. However, for design space with high dimension and large size, such way of exploration is not feasible, because it may take years to run. So, we use heuristics to prune down the design space and prioritize evaluation of more reasonable design points first. With heuristics, we are able to find a “good enough” design point with limited count of evaluation. Secondly, I learned how the design for width and scheduling affects the clock cycle time, power and energy. I also learned how cache size affect access energy and leakage. Generally, in-order processors with less fetch width will have less clock cycle time, less core leakage power and less energy per committed instruction. And machines with smaller cache size will have less access energy and less leakage.