Computer Architecture

Project 1 Report

A. Burak Gulhan 920177802
Agasthya Harekal 915750011

Design Space Exploration

The provided framework enables design space exploration regarding performance and EDP by altering parameters of the processor. We modify the configuration for each dimension, trying out all possible values for it and run a simulation for each these configurations to get the performance and EDP. We choose the value for this dimension that gives the best result (performance or EDP). We use a heuristic instead of checking all possible configurations. We determine the traversal order for the dimensions depending on our PSU id number (5^{th} category). We continue to check dimensions until we converge or reach 1000 design points.

Design Point Chosen by our Program

The best performance design point:

0 0 2 2 0 6 0 2 3 1 0 0 4 3 2 1 5 4

The best EDP design point:

0 0 2 2 0 5 0 1 3 1 0 0 4 3 1 1 4 3

TABLE OF PERFORMANCE AND EDP

Parameter	Performance	EDP
width	Value = 0 Why = Normally increasing ALUs would increase performance but the test code might not have enough ILP for this to matter	Value = 0 Why = Increasing instruction pipelines significantly increases energy consumption
scheduling	Value = 0 Why = In order execution doesn't have misspeculation which can decrease performance with a bad predictor	Value = 0 Why = misspeculation increases energy consumption
l1block	Value = 2 Why = Determines how much data can be read and written at the same time. Depending on how the memory is accessed in the code (in sequential order) then having large blocks may improve performance	Value = 2 Why = Accessing memory more frequently is more costly to EDP than the cost of having a larger il1 cache.
dl1sets	Value = 2 Why = Larger cache decreases miss chance and therefore decreases amount of time spent accessing memory. But there is a tradeoff, since larger caches have more latency.	Value = 2 Why = Accessing memory is more costly to EDP than the cost of having a dl1 cache.

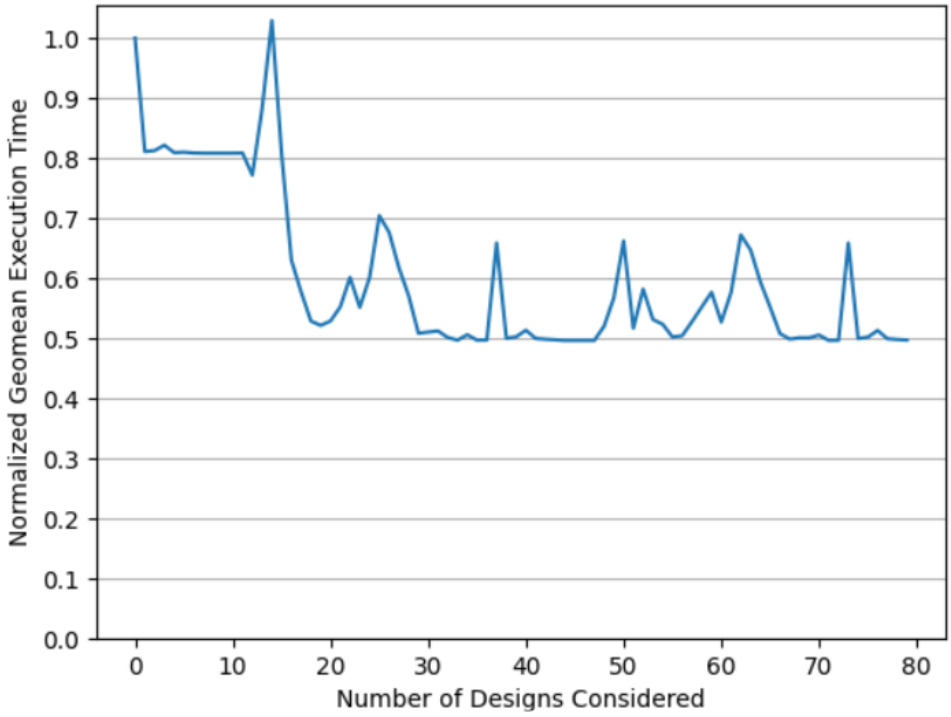
dl1assoc	Value = 0 Why = Higher associativity provides a higher hit rate but increases the placement time. If we have frequent cache placement, this might decrease performance.	Value = 0 Why = High associativity increases power consumption because of added hardware and the amount iterations needed for cache placement
il1sets	Value = 6 Why = We generally read instructions sequentially, depending on frequency of branches, caching instructions increase performance. Therefore, larger il1 cache decrease memory access for instructions.	Value = 5 Why = Accessing memory is more costly to EDP than the cost of having an il1 cache (up to a certain point, since larger caches use more power).
il1assoc	Value = 0 Why = Higher associativity provides a higher hit rate but increases the placement time. If we have frequent cache placement, this might decrease performance.	Value = 0 Why = High associativity increases power consumption because of added hardware and the amount iterations needed for cache placement
ul2sets	Value = 2 Why = Having a ul2 cache decreases chance of accessing memory on a cache miss.	Value = 1 Why = Accessing memory is more costly to EDP than having a ul2 hit, but having a larger

		ul2 also increases power consumption.
ul2block	Value= 3 Why = Same as il1block	Value= 3 Why = Same as il1block
ul2assoc	Value= 1 Why = Same as il1assoc	Value= 1 Why = Same as il1assoc
replacepolicy	Value= 0 Why = Using LRU for cache is more efficient than FIFO and random for our test code. Since LRU keeps track of how often each cache item was used.	Value= 0 Why = Using a better cache replacement policy decreases memory access which increases EDP.
fpwidth	Value = 0 Why = Depending on how much floating point operations we do, this value is chosen. In our test code we probably don't have much floating point operations since this was chosen as 0. Having a higher fpwidth increases latency.	Value = 0 Why = Floating point data increases latency and energy consumption. If we rarely have floating point operations a lower value is better.
branchsettings	Value = 4 Why = Using a combination branch predictor is gives better predictions, which in turn increases performance when there are branches.	Value = 4 Why = Having a better branch predictor decreases mispredictions, which increase power consumption.
ras	Value = 3	Value = 3

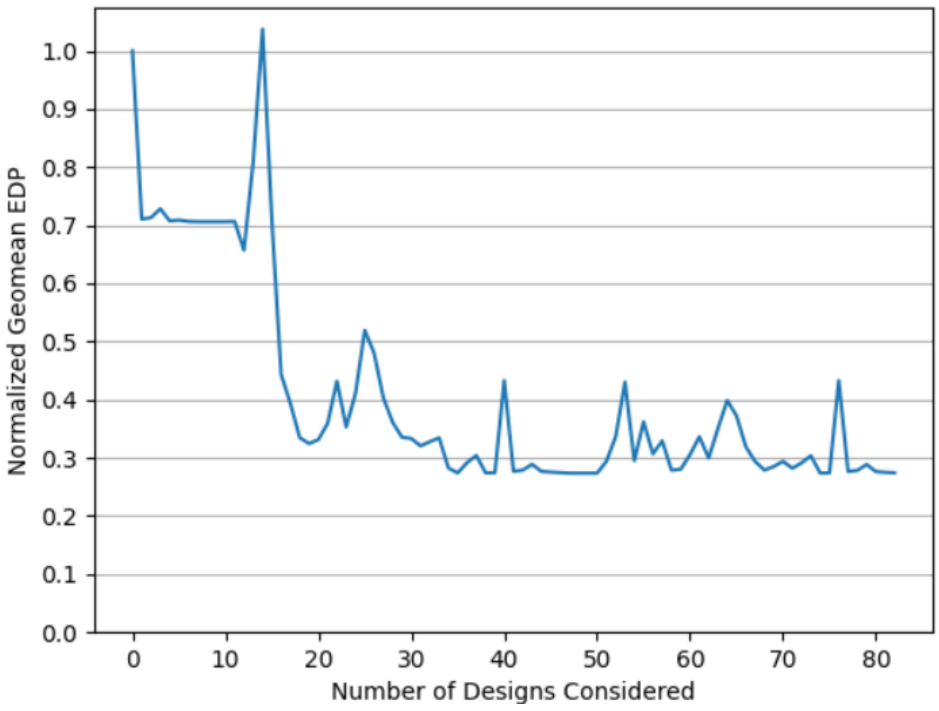
	Why = RAS stores return addresses for function calls, which prevent stalling when jumping back, which increases performance.	Why = Stalling decreases EDP
btb	Value = 2 Why = Depending on how many different branches we have, a larger btb is better.	Value = 1 Why = A larger BTB increases power consumption in each cycle. A smaller BTB might be more power efficient even if it decreases performance.
dl1lat	Value= 1 Why = This value is calculated depending on other values. It was not directly chosen by our heuristic.	Value= 1 Why = This value is calculated depending on other values. It was not directly chosen by our heuristic.
il1lat	Value= 5 Why = Same as dl1lat	Value= 4 Why = Same as dl1lat
ul2lat	Value= 4 Why = Same as dl1lat	Value= 3 Why = Same as dl1lat

Plots

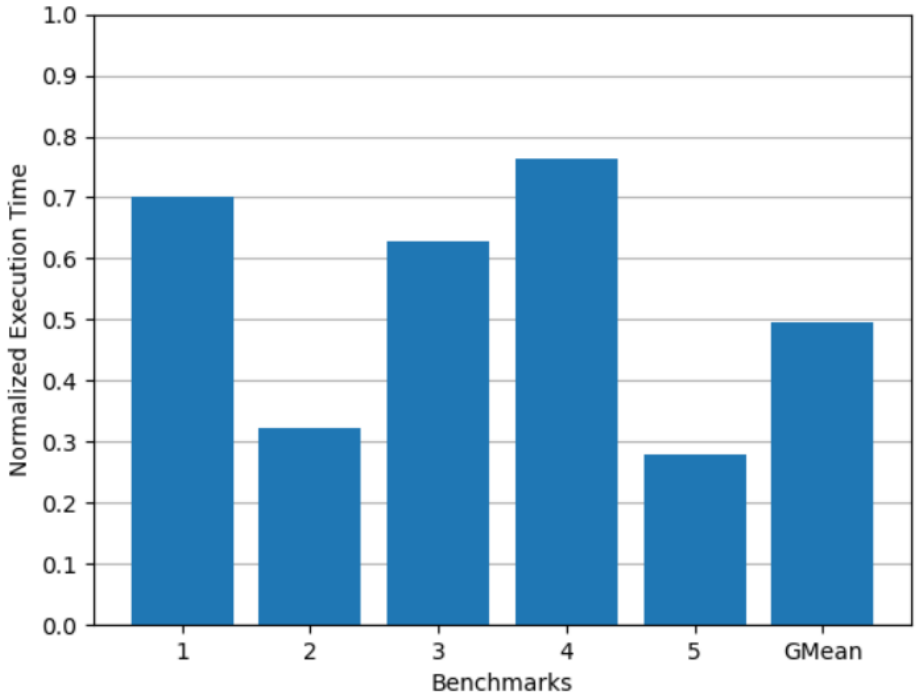
Plot 1:



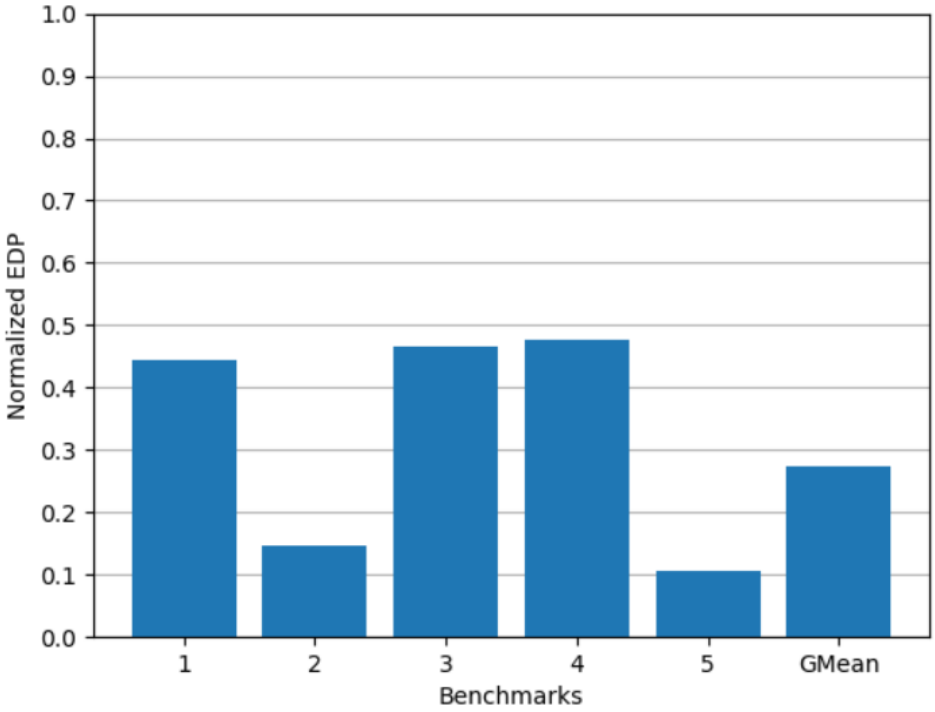
Plot 2:



Plot 3:


1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Normalized Execution Time
1
2
3
4
5
GMean
Benchmarks

Plot 4:


1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Normalized EDP
1
2
3
4
5
GMean
Benchmarks

More Sophisticated Heuristic Algorithm

Currently our heuristic takes around 100-200 design points to converge. Since we have 1000 design points available, we can use a more sophisticated heuristic. One problem in our heuristic is that we cannot temporarily decrease the performance for a larger performance increase later. We myopically set each dimension, but dimensions can also depend on other dimensions.

To mitigate this problem, we can change the order of our traversal categories each time we finish checking all dimensions. Since in our current heuristic we converge to a solution after about two passes over all dimensions (which takes around 100 design points). If we change our traversal order each time we finish checking all dimensions, then we can prevent the configuration from converging early and make use of our remaining design points.

There is still the issue of always traversing specific categories in the same order. We can mitigate this issue by changing the order of how we traverse each category. We can do this by reversing or randomizing the traversal order of the dimensions for the category.

We can further improve this heuristic by evaluating multiple dimensions at once, so that we pick the best configurations out of all possible configurations for only these multiple dimensions. This would significantly help with the issue discussed in the first paragraph. However, this would significantly increase the number of configurations tested depending on the dimension cardinality and number of dimensions evaluated at once. Therefore, the number of dimensions we evaluate once must be very small (at most 2 for 1000 design points).

Insights Gained

1) We learned about different design parameters of a CPU (the parameters on SimpleScalar) and how the performance and energy consumption of a CPU varies due to change in these parameters.
2) We learned what design space exploration is and how to explore the design space without checking every single permutation, by using a heuristic.

Additional Resources Used

Used the Python library, Matplotlib, to generate the plots.

Used online documentation regarding how SimpleScalar works and what each parameter means.

Additional Comments

We modified the 431project.h to include the functions getdl1size, getil1size, getl2size