Chase McFarlane & Andre Mitrik
CMPEN 431 – Project 1 Report

1. **Describe in 100 words or less how the provided framework and its components enable a design space exploration.**

SimpleScalar provides a model allowing virtual testing of components (e.g., CPU, Caches, and Memory Hierarchies) within an architecture to maximize performance based on criteria. Design space exploration is facilitated by given constraints about component parameters (e.g., sizes and associativity for caches) being used in conjunction with logical decision parameters (e.g., scheduling format and replacement policies) to determine the workflow for an optimized framework. These parameters are tested and calculated in a defined exploration order to fit specified criteria (performance vs energy optimization). Ultimately, the goal is to maximize the efficiency ratio of cache sizes to latency.

2. **List the design points chosen by your DSE.**

Optimized Performance Design Points: { 0 0 2 2 0 6 0 2 3 1 0 0 4 3 0 1 5 4 }

*DPI = Design Point Index

Parameter	***DPI**	**Value (unit)**	**Description**
Width	0	1 (word)	Only one instruction is issued per clock cycle
Scheduling	0	-issue:inorder true -issue: wrongpath false	Instructions are issued in order which keeps execution simpler when compared to out of order
L1 Block Size	2	32 (bytes)	The data cache blocks are 32 bytes in size (i.e., "wide")
L1 Data Sets	2	128 (sets)	The data cache is divided into 128 sets
L1 Data Associativity	0	1	The cache is 1-way associative which means each set only has one block of 32-bytes
L1 Instruction Sets	6	2048 (sets)	The instruction cache is divided into 2048 sets
L1 Instruction Associativity	0	1	The instruction cache is 1-way associative, meaning each set has one block 32-bytes
Unified L2 Sets	2	1024 (sets)	The unified L2 cache (instruction and data) is divided into 1024 sets
Unified L2 Block Size	3	128 (bytes)	The unified L2 cache is 128 bytes in size
Unified L2 Associativity	1	2	The unified L2 cache is 2-way associative, meaning each set has 2 blocks of 128-bytes (1 set = 256 bytes)
Cache & TLB Replacement Policy	0	l	The cache and TLB's blocks are replaced using a least recently used (LRU) policy on misses
Floating Point Unit Width	0	1	The floating-point unit has a width of 1
Branch Predictor Choice	4	-bpred comb -bpred:comb 1024	The branch predictor is a size of 1024 entries with a comb policy
Return Address Stack Size	3	8 (# entries)	The RAS contains 8 return address entries, meaning that 8 function calls' return addresses can be put into the stack
Branch Target Buffer	0	128 (# sets) 16 (associativity)	The BTB is divided into 128 sets with each set having 16 blocks of data (branch target addresses)
L1 D$ Latency	1	2 (time)	The data cache is the smallest of the caches which usually correlates to the lowest latency
L1 I$ Latency	5	6 (time)	The instruction cache is bigger than data cache
Unified L2 Latency	4	9 (time)	The unified L2 cache is the largest of the caches which evident by the highest latency

Optimized Energy Design Points: { 0 0 2 2 0 5 0 1 3 1 0 0 4 3 0 1 4 3 }

*DPI = Design Point Index

Parameter	***DPI**	**Value (unit)**	**Description**
Width	0	1 (word)	Only one instruction is issued per clock cycle
Scheduling	0	-issue:inorder true -issue: wrongpath false	Instructions are issued in order which keeps execution simpler when compared to out of order
L1 Block Size	2	32 (bytes)	The data cache blocks are 32 bytes in size (i.e., "wide")
L1 Data Sets	2	128 (sets)	The data cache is divided into 128 sets
L1 Data Associativity	0	1	The cache is 1-way associative which means each set only has one block of 32-bytes
L1 Instruction Sets	5	1024 (sets)	The instruction cache is divided into 2048 sets
L1 Instruction Associativity	0	1	The instruction cache is 1-way associative, meaning each set has one block 32-bytes
Unified L2 Sets	1	512 (sets)	The unified L2 cache (instruction and data) is divided into 1024 sets
Unified L2 Block Size	3	128 (bytes)	The unified L2 cache is 128 bytes in size
Unified L2 Associativity	1	2	The unified L2 cache is 2-way associative, meaning each set has 2 blocks of 128-bytes (1 set = 256 bytes)
Cache & TLB Replacement Policy	0	l	The cache and TLB's blocks are replaced using a least recently used (LRU) policy on misses
Floating Point Unit Width	0	1	The floating-point unit has a width of 1
Branch Predictor Choice	4	-bpred comb -bpred:comb 1024	The branch predictor is a size of 1024 entries with a comb policy
Return Address Stack Size	3	8 (# entries)	The RAS contains 8 return address entries, meaning that 8 function calls' return addresses can be put into the stack
Branch Target Buffer	0	128 (# sets) 16 (associativity)	The BTB is divided into 128 sets with each set having 16 blocks of data (branch target addresses)
L1 D$ Latency	1	2 (time)	The data cache is the smallest of the caches which usually correlates to the lowest latency
L1 I$ Latency	4	5 (time)	The instruction cache is bigger than data cache
Unified L2 Latency	3	8 (time)	The unified L2 cache is the largest of the caches which evident by the highest latency

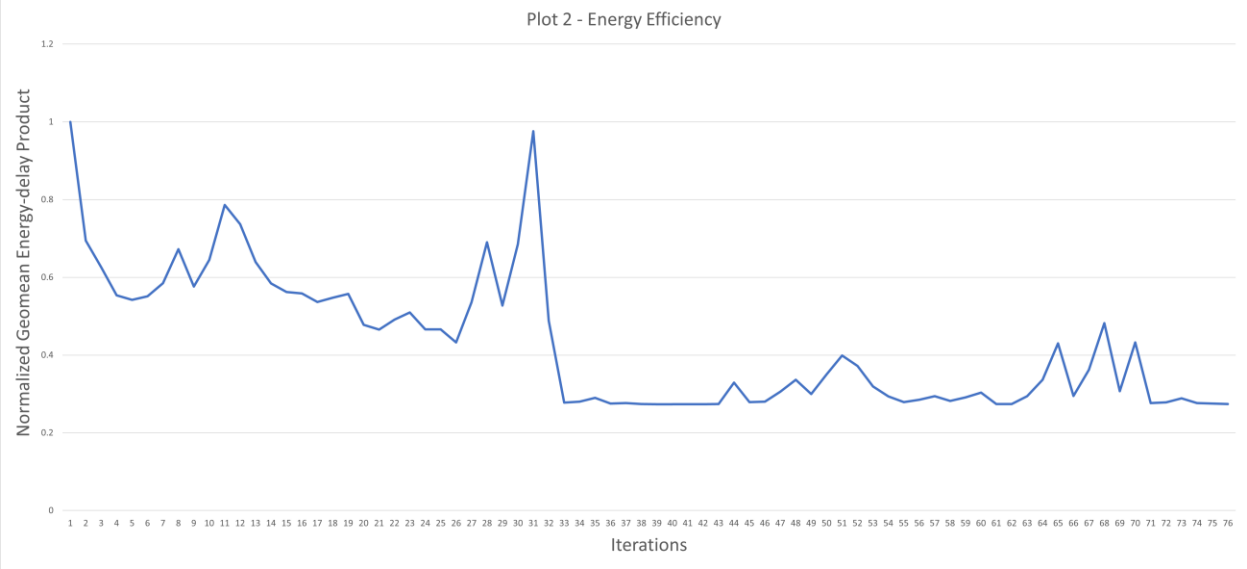
3. Fill out the following table as detailed below.

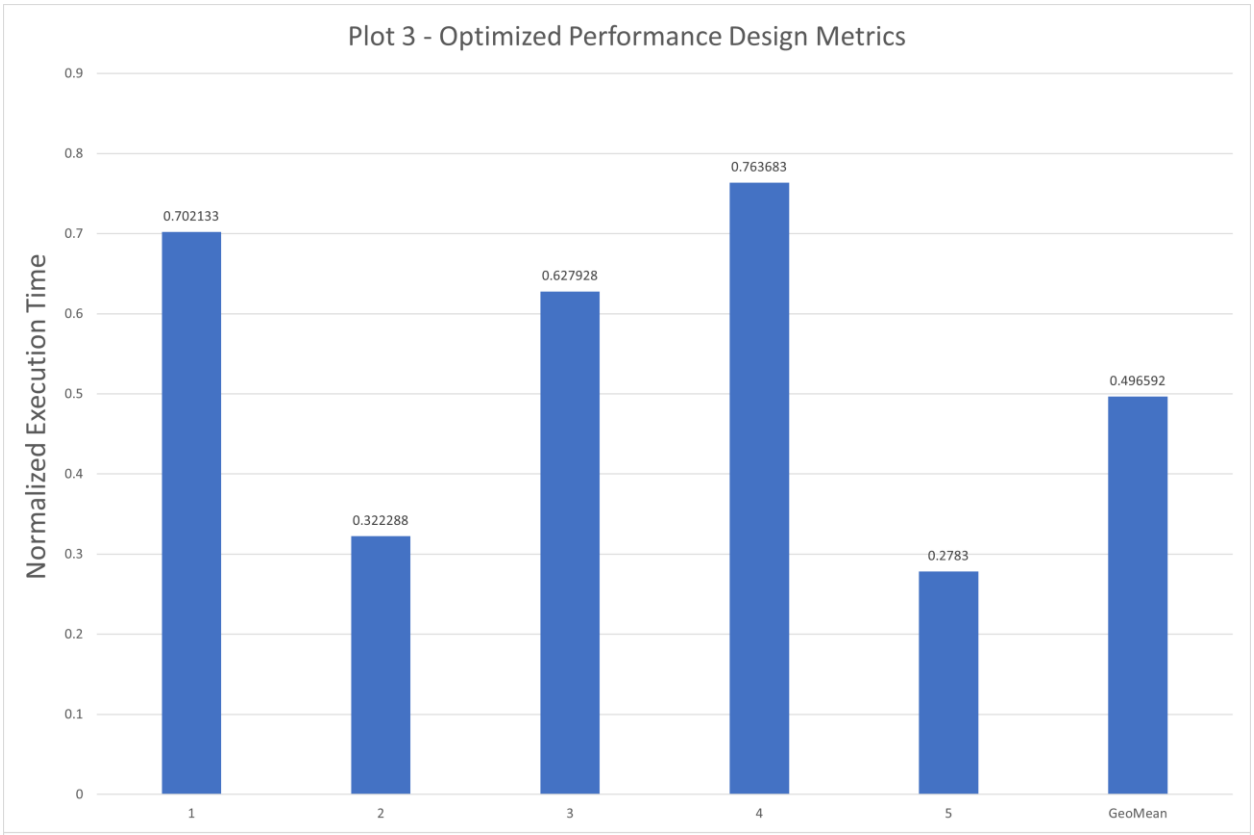
Parameter	Performance	EDP
Width	Value = 0 Why = 1 instruction issue is simpler and faster	Value = 0 Why = 1 instruction issue is simpler and requires less power (1 nW vs 2/4/8 mW)
Scheduling	Value = 0 Why = issuing in order reduces complexity thus increases issuing performance	Value = 0 Why = issuing in order reduces complexity and requires less power (1 mW vs 1.5 mW)
L1 Block Size	Value = 2 Why = 32-byte blocks are big enough such that sufficient information is held within each block without jeopardizing immediate performance	Value = 2 Why = 32-byte block sizes are large but permissible due to the frequency of access for L1 cache (these blocks also likely take significantly less power to manage)
L1 Data Sets	Value = 2 Why = 128 sets are appropriate for data since more sets should be devoted to instructions	Value = 2 Why = fewer sets for less frequently used parameters decrease overall cache size thus decrease power draw
L1 Data Associativity	Value = 0 Why = direct mapping only requires one comparator which proved to be more impactful than increasing hit rate	Value = 0 Why = 1-way only needs one comparator which requires less hardware compared to 2/4/8-way thus less power consumption
L1 Instruction Sets	Value = 6 Why = larger sets increase the cache size and hit rate thus increasing performance	Value = 5 Why = slightly smaller (compared to DSE perf.) because less sets to decrease total execution time
L1 Instruction Associativity	Value = 0 Why = direct mapping makes searches take less time and optimizes immediate performance	Value = 0 Why = direct mapping allows for easily findable instructions that requires the least expended machine energy
Unified L2 Sets	Value = 2 Why = 1024 sets should be enough size to not run out of memory soon while also maintaining access efficiency	Value = 1 Why = 512 sets is the minimum size that is big enough for proper functionality without stripping necessary performance
Unified L2 Block Size	Value = 3 Why = 128 bytes is permissible for an L2 cache since L2 caches feed the L1 caches (and	Value = 3 Why = 128 bytes is permissible because L2 caches are accessed infrequently (and will typically

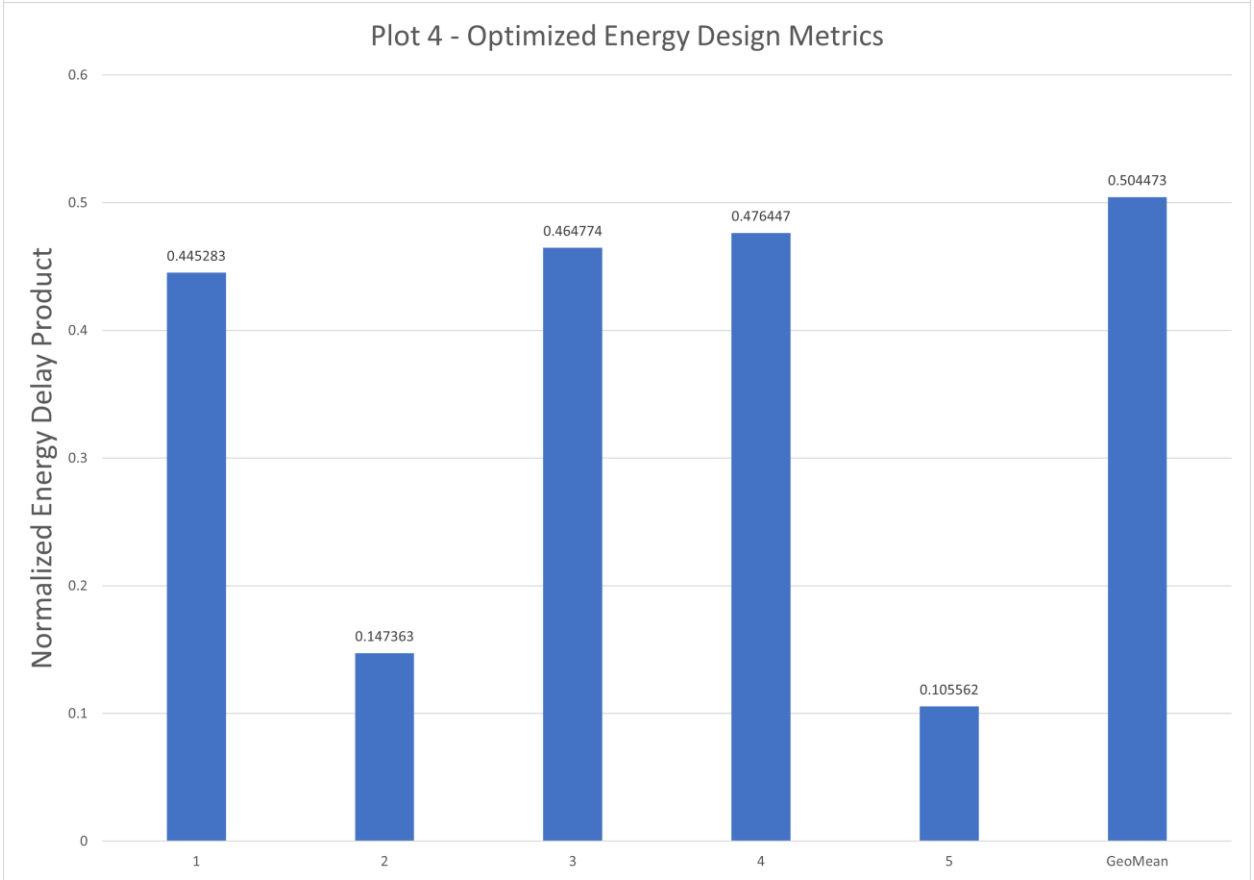
	hopefully do not need to be accessed often) and should not decrease performance	always take longer than L1 caches anyway)
Unified L2 Associativity	Value = 1 Why = 2-way associativity meets in the middle to store two blocks per set and access items via index and tag, maximizing performance within such a large block	Value = 1 Why = 2-way associativity should be the best format for a 128-byte L2 block to be able to quickly find required objects without spending too much time with indices or tags (one index, one tag per item)
Cache & TLB Replacement Policy	Value = 0 Why = LRU algorithms limit the time taken to access deeper caches of memory	Value = 0 Why = LRU algorithms only require temporal locality which is less stress on the machine's power output
Floating Point Unit Width	Value = 0 Why = FPU width directly determines FPU delay, and a width of 1 yields the smallest possible delay of 5 ps (best performance)	Value = 0 Why = FPU power is minimized with a width of 1 (0.25 mW)
Branch Predictor Choice	Value = 4 Why = A larger and more advanced branch predictor can increase accuracy which leads to less stalls thus increased performance	Value = 4 Why = A smaller branch predictor would draw less power but would be less accurate thus decreasing EDP. That is why SS determined a larger and more accurate branch predictor would be decrease EDP
Return Address Stack Size	Value = 3 Why = A larger RAS can maintain more return address entries (predictions) for when jump instructions are executed meaning increased prediction rates and performance	Value = 3 Why = A larger RAS allows for less power consumption when return address predictions are correct leading to decreased EDP
Branch Target Buffer	Value = 0 Why = 128 sets with 16-way associativity implies no required indices (since cache blocks can go anywhere) and allow for the most flexibility for performance	Value = 0 Why = Utilizing tags instead of indexes reduces the amount of energy required to find something if the BTB is used
L1 D$ Latency	Value = 1 Why = Product of previous parameters	Value = 1 Why = Product of previous parameters

L1 I$ Latency	Value = 5 Why = Product of previous parameters	Value = 4 Why = Product of previous parameters
Unified L2 Latency	Value = 4 Why = Product of previous parameters	Value = 3 Why = Product of previous parameters

4. **Plots as defined below.**




Plot 3 - Optimized Performance Design Metrics
Normalized Execution Time
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0.702133
0.322288
0.627928
0.763683
0.2783
0.496592
1
2
3
4
5
GeoMean


Plot 4 - Optimized Energy Design Metrics
Normalized Energy Delay Product
0.6
0.5
0.4
0.3
0.2
0.1
0
0.445283
0.147363
0.464774
0.476447
0.105562
0.504473
1
2
3
4
5
GeoMean

5. **Describe a more sophisticated heuristic which you expect will perform design space exploration (limited by 1000 design points) more effectively to find a better performing design (with respect to execution time).**

Different heuristics are situationally used to find optimized parameters based on a given goal for a project. Many heuristics exist (since they are just extended methodologies), but some specific heuristics include:

- Artificial Neural Networks (Machine Learning)
- Swarm Intelligence
- Simulated Annealing
- Support Vector Machines
- Genetic Algorithms

One such specialty that is growing in popularity is a neural network-based heuristic. Within this methodology, everything is based on a network of **neurons**. A neuron has the following properties:

1. Bias: the threshold in which a neuron "fires" for a certain value/criteria
2. Weight: a value defining whether the neuron under question is more important for the question at hand than others
3. Activation Function: transforms the weighted input of the neurons to the optimized configuration for the goal at hand

Neural networks also may use extremely precise calculations for the firing of neurons (e.g., ReLU function, Sigmoid activation function, Tanh activation function). For the sake of this example, we will use a sigmoid activation function. This function transforms the given inputs to a range between 0 and 1 for simplicity and easily readable results; these sets of results can be transformed into graphs to see trends within a network (for an overarching scope, or just for one variable). These graphs, when used in larger networks, give experimenters a much more reasonable interpretation of what each collection of neurons mean.

Neural networks also have traditional formats in terms of how they layer the neurons. Typically, they contain:

1. Input Layer: accepts the input data so that it can be branched through the entire network
2. Hidden Layers: the layers responsible for the inner complexity and calculations of the neural network itself. There can be as little as 1 or as many thousands of hidden layers depending on the degree of specificity for the neural network.
3. Output Layer: receiver of the output from the hidden layer to transform the data into a single result that gives an optimized result based on the preferences of the network

Given these terminologies, we can now define the problem at hand. When optimizing performance (execution time), we obviously want a compromise between the cache characteristics, efficiencies of the branch predictors, etc., and the execution time without sacrificing an extreme amount of performance. This requires an adaptive neural network that is trained through multiple iterations (not unlike the heuristic applied in this project. A basic operational flow can be defined below:

1. Generate instances for Neural Network (NN) to learn

    a. This would include a very basic (and probably inefficient) set of neuron layers that are preset to a default configuration (the parameter values found in any of the arrays within runprojectsuite.sh. These are the variables that change in weight and bias once the neural network becomes trained.
2. For the instance under question, a **feature vector** is required. The feature vector, in unison with the current optimal heuristic, can be normalized using some arbitrary algorithm and inserted into the training data. Add an iteration to the iteration counter and continue training the network.
3. Once these new parameter neuron values are extracted from the previous iteration, a layer (likely to be one or two at a time) will change in weights and biases for a more appropriate execution time. The cycle continues with the previous training to be cleared, except with a much higher degree of specificity.
4. An initial state (after the first run of the NN instance) will be run, leading to an adaptive heuristic search.
5. (2) is repeated with a new feature vector and a newer more optimized heuristic, and this data is implemented into another layer within the neural network. The cycle continues until the changes within the neural network are minimized.

Once the network is trained through repeated processes of finding optimal weights and biases within the hidden layers (e.g., determining which cache sizes have most significant jurisdiction on the subsequent layer's output), if required, hidden layers can be reordered and placed in a fashion such that the logical progression of the inputs optimizes the execution time.

6. **Elaborate on any 2 new insights you gained while working on this project.**

SimpleScalar is a simulator for computer architectures that can emulate the performance of an architecture without having to build the physical chips or components. This virtualization facilitates the host machine to keep operating under its current architecture (e.g., x86) while simulating another ISA (e.g., MIPS). It can be similarly compared to how virtual machines allow different operating systems to run on top of another unrelated operating system. Given this description of the overbearing architecture, two insights our group gained working on this project include:

1. The importance of virtualized simulator hardware components to save on R&D costs.
2. How to develop an effective way to approach a potential architecture's component design given a list of input parameters/constraints. Additionally, valuable insight was provided depicting the importance of the sequence of how these constraints are used and the effect the sequence has on the final output.

7. **List of additional resources used (optional).**

- TA Office Hours
- C++ math library (for log2 function)
- https://www.geeksforgeeks.org/c-plus-plus/
- http://www.simplescalar.com/
- https://en.wikipedia.org/wiki/Microarchitecture_simulation
- http://www.ecs.umass.edu/ece/koren/architecture/Simplescalar/Simulators.pdf
- https://utd-ir.tdl.org/bitstream/handle/10735.1/6256/ETD-5608-013-JAYASHEELGOWDA-8528.63.pdf?sequence=5&isAllowed=y
- https://web.cs.ucdavis.edu/~su/Berkeley/cs252/project.html
- https://projects.cerias.purdue.edu/stackghost/stackghost/node11.html
- https://www-classes.usc.edu/engr/ee-s/457/EE457_Classnotes/ee457_Branch_Prediction/EE560_05_Ras_Just_FYI.pdf

8. **Additional information or comments (optional)**

N/A