# CMPEN 431: Computer Architecture, Spring 2019

# Course Project #2
# Design Space Exploration

In this project, you are going to use `SimpleScalar` as the evaluation engine to perform a design space exploration, using the provided framework code, over a 18-dimensional processor pipeline and memory hierarchy design space (some of these dimensions are not independent). You will use a 5-benchmark suite as the workload.

## Project Goal

Your assignment is to, with an evaluation count limit of 1000 design points, explore the design space in order to select the best performing design under a set of **two** different optimization functions. These include:

1. the “best” performing overall design (in term of the geometric mean of normalized execution time normalized across all benchmarks)
2. the most energy-efficient design (as measured by the lowest geometric mean of normalized energy-delay product [units of energy delay product are joule-seconds] across all benchmarks)

## Background

### SimpleScalar

SimpleScalar is an architectural simulator which enables a study of how different processor and memory system parameters affect performance and energy efficiency. The simulator accepts a set of system design parameters and an executable (workload) to run on the described system. A wide range of system statistics are recorded by the simulator as the executable runs on the simulated system. Once the framework in this project is setup, interested readers can have a look at one of the log files in *`rawProjectOutputData`* folder to view SimpleScalar output.

This project heavily uses SimpleScalar but most of the interface is abstracted out by a simpler framework interface. Nevertheless, you can refer to this SimpleScalar guide for details about parameters passed to SimpleScalar.

### Design Space Exploration

Given a set of design parameters, Design Space Exploration (DSE) involves probing various design points to find the most suitable design to meet required goals. Follow this quick reading about DSE before moving ahead.

DSE can be performed for different design goals. For example, one DSE may want to find the best performing design whereas another DSE may be aimed at finding the most energy efficient design. A more complex DSE may look for the best performing design given a fixed energy budget.

An exhaustive DSE simply tries out all possible combinations of parameter values to find the absolute best design. However, as the size of design space increases this approach quickly becomes infeasible. Consider a 10-dimensional design space with 5 possible values for each parameter and 2 minutes simulation time to evaluate a given design point; an exhaustive search will take $5^{10}$ x 2 min ≈ 37 years.

A more intelligent DSE employs heuristics to intelligently prune down the design space and to prioritize evaluation of more reasonable design points first. If the assumptions employed by the heuristics are correct, the DSE will still result in the best design. On the other hand. with a set of reasonably justified assumptions a heuristic can result in a "good enough" design point.

### Energy-Delay Product

Energy-Delay Product (EDP) is a metric which consolidates both performance and energy efficiency.

$$EDP = total\ execution\ energy \times execution\ time$$

Design A takes 100pJ to process an image in 100ms, EDP = 10000 units. Design B takes 80pJ to process an image in 2000ms, EDP = 160000. Design A is clearly more energy efficient, but it performs poorly as it incurs more execution time. EDP enables a more holistic design comparison.

### OurSimpleHeuristic

We define OurSimpleHeuristic as follows:

- Design space dimensions can be labelled as either explored and unexplored.
- Initially all dimensions are unexplored
- For each dimension
    - Evaluate all possible design points by changing the value of this dimension only
    - Fix value of this dimension by selecting the best design so far (consider DSE goal)
    - Mark this dimension as explored

For simplicity, your implementation can select DSE dimensions starting from the left-most and moving rightwards OR in opposite direction depending on your PSU ID (more details later).

## Logistics

The set of possible points within the design space to be considered are constrained by the provided shell script wrapper `runprojectsuite.sh`. All allowed configuration parameters for each dimension of the design space are briefly described in the provided shell script.

`runprojectsuite.sh` shell script takes 18 integer arguments, one for each configuration dimension, which expresses the index of the selected parameter for each dimension. All reported results should be normalized against a baseline design with configuration parameters which already hard-coded in the framework.

Note that **not all possible parameter settings represent a valid combination**. One of your tasks will be to write a configuration validation function based upon restrictions described later in this document. Further, note that this design space is too large to efficiently search in an exhaustive manner. Hence, a heuristic will be developed to specify an order in which the design space will be explored.

The framework code will evaluate a fixed number of design points per run. **This parameter cannot be changed.** The key part of your task in this project is to implement a heuristic search function that selects the next design point to consider, given either a performance, or an energy efficiency goal. Note that the framework code must be run once for each of the optimization function options.

The framework, as given, provides functionality to enforce several, but by no means all, of the validation constraints. It is your job to implement validation functions to enforce constraints described throughout this document.

### Framework

A sample run to use the provided framework can look something like this:

```
# Extract project files archive and navigate to project directory.
make clean
make
./DSE performance
```

Different components of the framework are invoked in the following order:

`DSE (project binary) → runprojectsuite.sh (shell script) → SimpleScalar`

DSE binary invokes *`runprojectsuite.sh`* script which in turn invokes SimpleScalar simulator with appropriate arguments. Several log files are generated in project directory on every invocation.

## Anticipated Steps

These steps can serve as a high-level guideline to aid you during the project:

1. Enter *`MY_PSU_ID`* in *YOURCODEHERE`.c`*. This will activate a specific heuristic scanning order based on your PSU ID.
2. Setup provided framework to get a set of results using the provided "unintelligent" heuristic.
3. Implement *`validateConfiguration`* and *`generateCacheLatencyParams`* functions.
4. Implement OurSimpleHeuristic in *`generateNextConfigurationProposal`* for both optimization goals (a well performing design and an energy efficient design)
5. Complete Report

## Submission Requirements

**This is an individual project**. Submitted artifacts should include:

1. Project report
2. Code implementations of missing or stub functions within the provided framework

### Project Report

**Your report must at conform to requirements listed in Appendix A.** This report, data contained within and their analysis will be the primary means of assessing this project.

Your report must be submitted via Canvas. (PDF only)

### Code Implementations

You will submit the source files (Makefile, runprojectsuite.sh, *.*cpp* and *.*h)* of your implementation as a single *tar* archive for an audit of your implementation efforts. Ensure that your code compiles on CSE machines without errors. You can make changes to framework if you conclude that they are required. The following commands will be used to compile and execute your code (followed by analysis of generated log files):

```
# Extract project files archive and navigate to project directory.
make
./DSE performance
./DSE energy
```

## Modeling Considerations

The Instruction Count (IC) for each benchmark is a constant. Thus, for performance, you will be trying to optimize Instructions Per Cycle (IPC) and the Clock Cycle (CC) time. **Unless specified otherwise, the following modeling consideration have already been implemented in the framework to calculate EDP.** However, the provided information may be used for explaining design space exploration results.

### Clock Cycle Time

We will use the following very simplistic model for clock cycle time: The clock cycle time is determined by the fetch width and whether the machine is in-order, or dynamic as follows:

| | |
|---|---|
| • Dynamic, fetch width = 1: | 115 ps clock cycle |
| • In-order, fetch width = 1: | 100 ps clock cycle |
| • Dynamic, fetch width = 2: | 125 ps clock cycle |
| • In-order, fetch width = 2: | 120 ps clock cycle |
| • Dynamic, fetch width = 4: | 150 ps clock cycle |
| • In-order, fetch width = 4: | 140 ps clock cycle |
| • Dynamic, fetch width = 8: | 175 ps clock cycle |
| • In-order, fetch width = 8: | 165 ps clock cycle |

### Power and Energy

#### Core Leakage Power

| | |
|---|---|
| • Dynamic, fetch width = 1: | 1.5 mW |
| • In-order, fetch width = 1: | 1 mW |
| • Dynamic, fetch width = 2: | 2 mW |
| • In-order, fetch width = 2: | 1.5 mW |
| • Dynamic, fetch width = 4: | 8 mW |
| • In-order, fetch width = 4: | 7 mW |
| • Dynamic, fetch width = 8: | 32 mW |
| • In-order, fetch width = 8: | 30 mW |

#### Cache and Memory

Following list comprises tuples of format: [cache size or memory, access energy(pJ), leakage/refresh power(mW)]

- 8KB: 20pJ, 0.125mW
- 16KB: 28pJ, 0.25mW
- 32KB: 40pJ, 0.5mW
- 64KB: 56pJ, 1mW
- 128KB: 80pJ, 2mW
- 256KB: 112pJ, 4mW
- 512KB: 160pJ, 8mW
- 1024KB: 224pJ, 16mW
- 2048KB: 360pJ, 32mW
- Main Memory: 2nJ, 512mW

### Energy per Committed Instruction

| | |
|---|---|
| • Dynamic, fetch width = 1: | 10pJ |
| • In-order, fetch width = 1: | 8pJ |
| • Dynamic, fetch width = 2: | 12pJ |
| • In-order, fetch width = 2: | 10pJ |
| • Dynamic, fetch width = 4: | 18pJ |
| • In-order, fetch width = 4: | 14pJ |
| • Dynamic, fetch width = 8: | 27pJ |
| • In-order, fetch width = 8: | 20pJ |

### Validation Constraints

**You must implement these validation constraints in your code.** Specifically, *validateConfiguration* and *generateCacheLatencyParams* must be implemented properly.

1. The il1 (L1 instruction cache) block size must match the ifq (instruction fetch queue) size (e.g., for the baseline machine the ifqsize is set to 1 word (8B) then the il1 block size is also set to 8B). The dl1 (L1 data cache) should have the same block size as your il1.
2. The ul2 (unified L2 cache) block size must be at least twice your il1 (and dl1) block size with a maximum block size of 128B. Your ul2 must be at least twice as large as il1+dl1 in order to be inclusive.
3. il1 size and dl1 size: Minimum = 2 KB; Maximum = 64 KB
4. ul2 size: Minimum = 32 KB; Maximum = 1 MB
5. The il1 sizes and il1 latencies are linked as follows (the same linkages hold for the dl1 size and dl1 latency):
   - il1 = 2 KB means il1lat = 1
   - il1 = 4 KB means il1lat = 2
   - il1 = 8 KB means il1lat = 3
   - il1 = 16 KB means il1lat = 4
   - il1 = 32 KB means il1lat = 5
   - il1 = 64 KB means il1lat = 6
   - The above are for direct mapped caches. For 2-way set associative add 1 additional cycle of latency to each of the above; for 4-way add 2 additional cycles.

6. The ul2 sizes and ul2 latencies are linked as follows:
    - ul2 = 32 KB means ul2lat = 5
    - ul2 = 64 KB means ul2lat = 6
    - ul2 = 128 KB means ul2lat = 7
    - ul2 = 256 KB means ul2lat = 8
    - ul2 = 512 KB means ul2 lat = 9
    - ul2 = 1024 KB (1 MB) means ul2lat = 10
    - The above are for direct mapped caches. For 2-way set associative add 1 additional cycle of latency to each of the above; for 4-way add 2 additional cycles; for 8-way add 3 additional cycles; for 16-way add 4 additional cycles.

## Miscellaneous Constraints

These constraints have already been specified in the framework. Have a look at SimpleScalar invocation command in *`runprojectsuite.sh`* for an exhaustive list of specified parameters. Moreover, any parameter not specified in *`runprojectsuite.sh`* will default to SimpleScalar default settings.

- mplat is fixed at 3
- fetch:speed is fixed at 1
- ifqsize can be set to a maximum of 8 words (64B)
- decode:width and issue:width equal to your fetch:ifqsize
- mem:width is fixed at 8B (memory bus width)
- memport if fixed at 1
- mem:lat is fixed at 51 + 7 cycles for 8 word
- tlb:lat is fixed at 30, maximum tlb size of 512 entries for a 4-way set associative tlb

# Appendix A

## Project Report Minimum Requirements

Your report must at least answer the following prompts in the exact same order.

1. Describe in 100 words or less how the provided framework and its components enable a design space exploration.
2. List the design point chosen by your DSE.
3. Fill out the following table as detailed below.
4. Plots as detailed below
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).
6. Elaborate on any 2 new insights you gained while working on this project.
7. List of additional resources used (optional)
8. Additional information or comments (optional)

### Table

In each cell specify the parameter value followed by why this value guided the DSE closer to your optimization goal (for example: more ALUs allow extraction of more ILP and increase performance). Make sure the parameters are in the exact order as they appear in *`runprojectsuite.sh`*

| **Parameter** | **Performance** | **EDP** |
|---|---|---|
| **Param1 (i.e. width)** | Value =<br>Why = | Value =<br>Why = |
| **Param2 (i.e. scheduling)** | Value =<br>Why = | Value =<br>Why = |
| **…** | … | … |
| **ParamN** | Value =<br>Why = | Value =<br>Why = |

### Plots

The report should include the following four plots:

A. Line plot of normalized geomean execution time (y axis) for each considered design point vs. number of designs considered (x axis)
B. Line plot of normalized geomean of energy-delay product (y axis) vs number of designs considered
C. Bar chart showing normalized per-benchmark 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

**These four plots must be labelled in your report corresponding exactly to numbering in the list above**. Furthermore, axis in the plots should be properly labelled

## Other Guidelines

For clarity in the written report, when listing the best design points, please do not represent them in terms of their index representations (e.g. 1 0 0 5 2 ...) and instead describe the actual value used for each dimension in a table or similar presentation.

Points will also be assigned for following the guidelines and adhering to appropriate levels of clarity, and style (and spelling, grammar, etc.) for a technical document.

# Appendix B

## Project FAQs

**Q: What are the column headers for the .log file?**

A: normalized EDP, normalized Execution time, absolute EDP, absolute Execution time

The writes to both the .best and .log files are generated near the end of main.

**Q: What are the column headers for the .best file?**

A: Headers differ by line:

**Line 1 headers**: bestEDPconfig, normalized EDP of bestEDPconfig, normalized Execution time of bestEDPconfig, absolute EDP of bestEDPconfig, absolute Execution time of bestEDPconfig, absolute EDP of Bench 0 on bestEDPconfig, normalized EDP of Bench 0 on bestEDPconfig, absolute EDP of Bench 1 on bestEDPconfig, normalized EDP of Bench 1 on bestEDPconfig, absolute EDP of Bench 2 on bestEDPconfig, normalized EDP of Bench 2 on bestEDPconfig, absolute EDP of Bench 3 on bestEDPconfig, normalized EDP of Bench 3 on bestEDPconfig, absolute EDP of Bench 4 on bestEDPconfig, normalized EDP of Bench 4 on bestEDPconfig

**Line 2 headers**: bestTimeconfig, normalized EDP of bestTimeconfig, normalized Execution time of bestTimeconfig, absolute EDP of bestTimeconfig, absolute Execution time of bestTimeconfig, absolute Time of Bench 0 on bestTimeconfig, normalized Time of Bench 0 on bestTimeconfig, absolute Time of Bench 1 on bestTimeconfig, normalized Time of Bench 1 on bestTimeconfig, absolute Time of Bench 2 on bestTimeconfig, normalized Time of Bench 2 on bestTimeconfig, absolute Time of Bench 3 on bestTimeconfig, normalized Time of Bench 3 on bestTimeconfig, absolute Time of Bench 4 on bestTimeconfig, normalized Time of Bench 4 on bestTimeconfig

**Q: Why are there only 18 configuration parameters when SimpleScalar (and the project specification) list so many more?**

A: There are 18 configuration variables, and more derived settings from those 18 configuration variables, and still more settings that are fixed as constant (e.g. MPLAT). Given the block size (set independently), associativity (set independently), and number of sets (set independently), you can determine total cache size for the L1 D and I caches and then validate if the latency for that cache (set independently) is set correctly.

**Q: What's a quota error, why are half my output files empty, and why can't I make new files anymore?**

A: It means you are out of disk space. Each run of this program produces a large number of intermediate output files for the evaluated design points. These are kept to speed up subsequent evaluations of the same design point in future runs as a means of reducing debugging/heuristic development time. Consider cleaning out your browser caches if you are low on disk quota before performing a project run.