# Project 2 : Two-Level Performance

## 1. Introduction

To hide main-memory latency, modern CPUs employ a hierarchy of caches. In this project you will build a two-level, direct-mapped data-cache simulator, incorporate miss penalties, and quantify overall performance via effective CPI.

## 2. Two-Level Cache Simulator

Develop a cache simulator using the following memory hierarchy description. The Simulator must utilize the given memory pattern generators.

### 2.1. Cache Hierarchy

L1 Cache

- Size: 16 KB
- Line size: 16 B, 32 B, 64 B, 128 B (variable)
- Associativity: 4-ways
- Hit time: 1 cycle

L2 Cache

- Size: 128 KB
- Line size: 64 B (fixed)
- Associativity: 8-way
- Hit time: 10 cycles

Main Memory (DRAM)

- Size: 64GB
- Access penalty: 50 cycles

### 2.2. Assumptions

- 35% of the instructions are loads and stores. These are the instructions that use the memory hierachy. 50% of the memory accesses are memory read.
- The instructions are fetched from an ideal memory.
- The caches employ the write-back algorithms
- Random replacement is used
- The CPU CPI is 1 when using an ideal caching system (100% hit in L1 cache)

### 2.3 The Simulator Skeleton

```
cycles = 0
for 1M iterations {
        p = random value from 0 to 1;
        if(p<=0.35){
                call the memory generator to get an address;
                rdwr = random value from 0 to 1;
                if(rdwr < 0.5) memory_access_type = READ;
                else memory_access_type = WRITE;
                pass the address to L1 cache;
                if(hits in L1) cycle++;
                else {
                        cycle = cycle + 1 + no. of cycles to get the data from the next level;
                }
        } else cycle++;
}
CPI = cycles / 1M;
```

### 2.4 Data Collection

Run the simulator using different generators (5 of them) and different L1 line size (4 sizes) and report the average CPI. This involves 20 simulation runs. Graph the CPI vs the line size fro different memory generators. Analyze the graphed data and draw some conclusions.

## 3. Deliverables

- The cache simulator source code (on GH)
- A report (2 pages in addition to the cover) to present the collected data and your analysis. The analysis involves plotting the collected data and outlining the conclusions that can be extracted from the graphed data.
- You must submit your report and source code as well as schedule an appointment for the demo on July 22nd. More information to follow.

## 4. Grading

- Simulator implementation with test cases used to verify its functionality [40%].
- Experiments execution, data collection, and data presentation (report + Interview) [30%].
- Data analysis and conclusions (report + Interview) [30%].