Lab 2: Implement a memory hierarchy

In this lab you will implement in an architectural simulation a memory hierarchy with multiple levels of cache. You will use the Pin-based starter simulator provided as base code and you will implement progressively more sophisticated memory hierarchy components in this simulator framework.

Learning Goals:

- Understanding simulator implementation and the implementability of simulated architectural features
- Understanding the design and implementation of set associative caches
- Understanding the design and implementation of multi-level cache hierarchies
- Understanding the implementation of a variety of cache replacement policies.
- Understanding how to use a simulator's output with varied configurations to see the impact on performance
- Using Average Memory Access Time (AMAT) to perform an iterative design space search to find an optimal performance cache hierarchy with minimized power and area costs.

Task 1:

Implement a set associative cache. Your cache should take a variable power-of-two associativity (up to a reasonable maximum), variable block size, and variable total size.

Evaluation: Measure the miss rate of your cache with different levels of associativity and block size. Create a plot showing the average memory access time at each level of associativity and block size.

```
AMAT (1-level cache) = L1_hit_rate * L1_access_cost + L1_miss_rate *
                     ( L1_miss_cost + DRAM_access_cost)
```

Task 2:

Implement the random, LRU, and bit-PLRU replacement policies from class, and any other replacement policy that you think might improve performance.

Evaluation: Measure the difference in miss rate with each policy. Create a plot showing the miss rate for each of the replacement policies for three different, reasonable cache size/associativity configurations. Observe the presence or absence of a trend in these data. Argue in writing that your size and associativity configuration choices are reasonable.

Task 3:

Implement a two-level cache hierarchy with parameterized cache size, block size, set associativity, and replacement policy at each level.

Task 4:

Write a design space iteration tool that searches the space of possible cache hierarchy configurations. A configuration is a tuple consisting of

```
((L1 associativity, L1 block size, L1 size, L1 replacement),
 (L2 associativity, L2 block size, L2 size, L2 replacement))
```

Your design space iteration tool should search the space of configuration tuples to find the optimal configuration. To evaluate a configuration, your design space iteration tool should run the set of test programs for that configuration. You should evaluate each cache's leakage power, dynamic access power, and access latency using the Destiny memory modeling tool. Using these numbers from Destiny, you should report a configuration's performance as the Average Memory Access Time (AMAT). For DRAM latency, you should use DRAM-4000 Column Address Strobe latency, which is 19 cycles at the DRAM's clock speed, which has a clock period of 0.5ns; DRAM latency is 9.5ns.

```
AMAT (2-level cache) = L1_hit_rate * L1_access_cost + L1_miss_rate * ( L1_miss_cost +
         L2_hit_rate * L2_access_cost + L2_miss_rate * ( L2_miss_cost +
             DRAM_access_cost))
```

Total Storage Budget:

Your system has a total budget of 5MB of cache that you can split across the layers of cache in your system. Your design space iteration tool should limit its search to cache hierarchies under this total amount of cache storage.

Minimization goal:

Your design space iteration tool should minimize area and power if comparable configurations are equal in their performance; always opt for the lower area or lower power configuration.

Implementability:

You must argue in your write-up that the cache hierarchy that you have proposed is implementable using supporting evidence from Destiny. Is the power consumed while doing reads and writes reasonable? Is the tag storage overhead reasonable? Are access latencies reasonable? Your argument should consider that L1 accesses should be only a few cycles, meaning L1 access latency (especially for reads) is critical.

Evaluation & Submission Artifacts:

You will submit your cache simulator, your design space search scripts, and a coherent README that is sufficiently detailed for your TA to run your simulator on other test workloads.

- You will submit a separate document (pdf) that is a writeup of your cache simulator.
- You should describe any important design choices that you made in your simulator design. You should answer any direct questions posed to you in this handout.

- You should include a quantitative summary of your design space search, including a design space plot including a Pareto Frontier cataloging the set of design points that you considered in your design space search.
- You should include a bar plot for a set of Pareto optimal cache configurations that shows the AMAT for each of the SPEC2017 benchmarks that you run.
- You should include a sensitivity study that shows how performance varies with the important dimensions of your cache. Plot performance of your benchmarks as L2 associativity increases. Plot performance of your benchmarks as L2 size increases. You may have to do some exploration to identify dimensions of your design along which you find interesting variations in performance.

Tips: You should provision a substantial amount of time for data analysis, running additional experiments, and plotting your results. Coding and getting the experiments to run is barely "half of the battle" in this lab. The design space exploration, design iteration, data analysis, sensitivity studies, and plotting are also a major part of the work of this lab. Be careful with your data and consider making timestamped and "configuration-stamped" directories that store data from different experiments. These considerations will allow you to go back an revisit and experiment's data later.