Project Report

Implementation

Our implementation of the cache simulator begins with **main.cpp**, which handles all input / output, containing the components of the L1 cache, L2 cache, DRAM, and our metrics tracker. In the directory, the **traces** folder is provided to use for benchmarking, while **outputs** is used for summaries of energy / time consumption per file and set associativity, and **logs** for more detailed information about cache hits / misses per run.

L1 is implemented as 2 direct-mapped caches, each with 512 entries consisting of an address, 2 bits for tracking validity and dirtiness of an entry (we could have fit this inside the address but moved it outside for readability), and the 64B line of values. The address is split into the tag, index, and offset per standard for the capacity of 512, and interactions for reading / writing values are facilitated by helper functions to read and write entire lines in the cache after byte alignment. In particular, data writes are implemented by reading the line, replacing a designated section, and writing back the line. This is implemented as a write-back cache (instead of write-through), where we only write to L2 upon eviction, due to prior specification. Furthermore, it is important to note that the L1 level has deterministic eviction policies due to the implementation of direct-mapping, which means that metrics at this level should be consistent through runs (which we find in our metrics).

L2 is implemented as a combined cache, all found in the **l2.h** file, due to compile-time templating for changing set associativity with structs. Here, we vary the address that we store in each entry in a set based on the associativity, with (4096 / set associativity) total sets. To determine which entry in a set is evicted, we sample from a random uniform distribution, which can be seeded (at **l2.h:49**) for pseudodeterministic output in testing. Other than this, this level behaves similarly to the L1 level (which are inclusive), where we only use the *L2::writeLine* and *L2::readLine* functions to interact with data, using write-back policies to update DRAM upon eviction.

DRAM is implemented as a hashmap of lines, with a theoretical bound of 8GB (but not tracked since we are allowed to assume all L2 misses can be found here).

To track metrics, we encode our constants defined by the original specification in **metrics.h**. Whenever we perform any reads at a level, we will call *Metrics::step* which modifies our metrics. This is calculated by computing the total idle (static) power for all caches and multiplying by the time we take for the read at the given level, combined with penalties and additive energy for the active level itself. Since it is specified that all writes are asynchronous (and we don't implement write-through between L1 and L2, only write-back), we only track metrics for reads and active energy for writebacks (we assume penalties are not tracked for writes). We can assume we are allowed to do this because in

the 0.5ns of a writeback to L2 or DRAM, the next instruction will still be in the L1 stage, which means we will not duplicate active energy at any level in this timeframe. With regards to clock speed, we also make the assumption that the instruction fetch, decode, and other non-memory-affiliated stages are instantaneous (inspired by this post, a L1 read hit would be 0.5ns in our implementation), so only L2 and DRAM levels will cause the simulator to stall.

Sample Metrics and Tables

Here, we have provided some samples for the **047.tomcatv.din** benchmark, with the summary consumption results based on set associativity from **outputs** and one run for details about cache hits and misses from **logs**. More metrics can be found in these folders for each benchmark. Since our policy in L2 eviction is non-deterministic, these metrics may vary slightly per simulation, but can be adjusted to use a seed for random generation. Tables are generated manually using data sampled from **logs** and **outputs**. Since we've already generated these metrics and submitted prior to the announcement (and want to keep below samples the same as those in our submission), it is important to note that we do not have a specific section for DRAM accesses in **logs**, but these are conceptually the same numbers as L2 misses - we have included modifications in our code so that future runs will have this section generated.

```
----------------------------------------------------------------
Simulation Results for Set Associativity 2:
----------------------------------------------------------------
Mean Energy (J): 0.00359442
StDev Energy (J): 4.35367e-06
Mean L1 Energy (J): 0.00715
Mean L2 Energy (J): 0.000879234
Mean DRAM Energy (J): 0.0117404

Mean Time (s): 0.000905432
StDev Time (s): 8.17178e-07
Mean Time per Instruction (ns): 0.905432

----------------------------------------------------------------
Simulation Results for Set Associativity 4:
----------------------------------------------------------------
Mean Energy (J): 0.0035881
StDev Energy (J): 2.89209e-06
Mean L1 Energy (J): 0.00715
Mean L2 Energy (J): 0.000879234
Mean DRAM Energy (J): 0.0117091

Mean Time (s): 0.000904212
StDev Time (s): 5.39592e-07
Mean Time per Instruction (ns): 0.904212

----------------------------------------------------------------
Simulation Results for Set Associativity 8:
----------------------------------------------------------------
Mean Energy (J): 0.00358764
StDev Energy (J): 1.91674e-06
Mean L1 Energy (J): 0.00715
Mean L2 Energy (J): 0.000879234
Mean DRAM Energy (J): 0.0117006

Mean Time (s): 0.000904108
StDev Time (s): 3.55046e-07
Mean Time per Instruction (ns): 0.904108
```

```
----------------------------------------------------------------
Set Associativity 2, Run 1:
Total Energy (J): 0.00359849
Total Time (s): 0.000906196
----------------------------------------------------------------
Total Instructions Executed: 1000000
Reads (0): 253723
Writes (1): 130733
Fetches (2): 615544
----------------------------------------------------------------
L1 Instruction Cache Fetches: 615544
L1 Instruction Cache Misses: 88
----------------------------------------------------------------
L1 Data Cache Fetches: 384456
L1 Data Cache Misses (Read): 1281
L1 Data Cache Misses (Write): 9040
----------------------------------------------------------------
L2 Cache Fetches (Instruction): 88
L2 Cache Fetches (Read): 1281
L2 Cache Fetches (Write): 9040
L2 Cache Misses (Instruction): 81
L2 Cache Misses (Read): 184
L2 Cache Misses (Write): 8548
----------------------------------------------------------------
```

	Set Associativity		
	2	4	8
Mean Energy	0.00359442 J	0.0035881 J	0.00358764 J
StDev Energy	4.35367e-06 J	2.89209e-06 J	1.91674e-06 J
Mean Time	0.000905432 s	0.000904212 s	0.000904108 s
StDev Time	8.17178e-07 s	5.39592e-07 s	3.55046e-07 s

Set Associativity			
	2	4	8
L2 Cache Fetches (Instruction)	88	88	88
L2 Cache Fetches (Read)	1281	1281	1281
L2 Cache Fetches (Write)	9040	9040	9040
L2 Cache Misses (Instruction)	81	81	81
L2 Cache Misses (Read)	184	178	145
L2 Cache Misses (Write)	8548	8529	8528

Set Associativity Analysis

In the above metrics, we choose the **tomcatv** benchmark due to its length and multitude of L2 cache misses, resulting in random evictions at the L2 level, which allows us to assess the impact of set associativity on energy and time consumption.

Among the different benchmarks, it is hard to determine which trends may be significant as a result of the difference in L2 random eviction - many of the results are very close together, and as indicated by this post, some benchmarks may not entail a significant amount of writeback to DRAM. For the given **tomcatv** benchmark, as well as many other benchmarks, we notice a downward trend in energy and time consumption with increasing set associativity. This generally makes sense given the context of our energy and time consumption calculations, since higher set associativities allow data to be more flexibly placed in sets, resulting in less cache misses and subsequently, less accesses to DRAM for these benchmark operation sets. However, we also notice a trend of higher energy and time consumption with increasing set associativity for some benchmarks, such as **compress** and **hydro2d**. We suspect that these trends may be a result of higher set associativities grouping indices together in the same set, where random L2 evictions may affect operations that have close addresses more than they would in a direct mapped cache (especially since we use random eviction rather than an adaptive algorithm like LRU).