2410 Project 2 Writeup/Design Description
Authors: Nathan Ackerman, Bin Dong, Paul Elder

How the heck does it work?

Compiling and Running:
To compile, you simply need to run “javac *.java” from within the src folder.
The command to run is “java CacheSimulator path_to_config path_to_trace”.
Adding a third argument will cause the program to run in debug mode and print information about what is happening, i.e. “java CacheSimulator path_to_config path_to_trace 1”

Config Files:
Config files are of the following format

p=2
n1=10
n2=10
a=2
b=8
dc=20
dm=100

Preprocessing the trace file:
To make things easier, we preprocess the trace file which allows us to split the requests into queues for each core. Moreover, we implemented them as Delta Queues to ease in managing the relative nature of them. We only ever need to decrement the time until the next issued instruction of the first element of the queue and then remove it once it is completed, since there is no out of order execution.

The Cores:
Each core has a private L1 cache and a piece of the shared distributed L2 cache which we denote L2piece. Each of these are subclasses of the cache class which override some functionality like handling evictions. For example, L1 caches on evict need to remove themselves from the list of people using that block (in the directory) and L2 evictions remove that directory entry entirely. There is a directory at each core that holds entries for each of its blocks but this is handled by the L2Arbiter. The cores also have a queue to fulfill requests for updating entries for control messages and it can update up to 1 entry per cycle in its L1. On each cycle, if the head request in the delta queue is ready, it is issued and the request goes through the caches. These calls are non-blocking and the core which check on each cycle for when that request is completed and compute the total time it took based on the difference between when it was issued and the current cycle.

L1 Cache:
The L1 cache is an extension of the cache class. Its entries were updated to include the MSI states. Upon a miss, the request is sent to the appropriate L2 piece through the L2Arbiter. Even on a hit the L2Arbiter may be used if it was a write and the block was not already in an exclusive state.

L2 Arbiter:
This is the main component of enforcing coherence. Additionally it enforces the communication limitations of the interconnect, in that only one processor can talk to a given processor on a specific cycle. It does this through the use of queues. When a request comes in, its address is used to map it to the core which would have that piece of data if it is in the L2. We assumed that since the directories are static RAM, similar to the caches, and since there is one directory per cache block, the sizes of the L2 cache piece and directory piece are similar and so they probably have the same latency, so we assume a directory lookup is also dc cycles. Requests for data are enqueued at the necessary core and the arbiter enforces that a core can only receive one communication per cycle. When a data request is processed, it checks the L2 cache for the data. If it is there is checks the directory and sends out the necessary control signals. It waits for responses and in the case that someone had it exclusive in their L1 it will wait for the new data. The requests is moved to a list of pending requests and once all the responses have been resolved the block will be given to the requesting core. If the L2 misses, the requests goes to a queue of memory requests and after dm cycles it is resolved and the directory is made and the block is put into the L2 and sent to the requesting core to be put in its L1.

Comparative Analysis

From Table 1 we can see that the size of the L1 cache and the associativity had no effect on any of the metrics. This is odd, as one would expect better performance in general with larger caches. It could also be that the traces were not formed in such a way that they could take advantage of larger caches. The metrics are realistic, however, in that the metrics for Trace 2 are an order of magnitude less than Trace 1's, since Trace 2 was an order of magnitude shorter than Trace 1.
Also, due to how the simulation works by allocating arrays of arrays of cache blocks for every processor core, we ran out of memory when doubling the L2 cache size (to 128KB per L2 tile). Doubling the L1 cache size worked fine, but we ran out of memory when we multiplied the L1 cache size by 16 (from 8KB per core).

Table 1:

	Total cycles	L1 misses	Total L1 miss time (cycles)	Avg L1 miss time (cycles)	Data messages	Control messages
Trace 1 reg	108614	2889	80296	27.79	5779	147
Trace 1 double L1	108614	2889	80296	27.79	5779	147
Trace 1 double L2	Out of memory					
Trace 1 double associativity	108614	2889	80296	27.79	5779	147
Trace 2 reg	14124	246	6760	27.48	492	26
Trace 2 double L1	14124	246	6760	27.48	492	26
Trace 2 double L2	Out of memory					
Trace 2 double associativity	14124	246	6760	27.48	492	26