CS622: Advanced Computer Architecture **(Due: 6th October 2020)**

Assignment 1: Cache Simulation

Instructor: Prof. Mainak Chaudhuri *Group 21:* Shobhit Jagga and Anish Saxena

Code Execution

Extract the zip file to obtain `group21/` directory. Place the traces in `traces/` directory. To simulate all traces for all cases in Part 1 and Part 2, follow the steps:

```
group21/$ make                  (compilation)
group21/$ ./run_all_traces.sh   (dumps output in results/ directory)
```

The cache simulator program, `driver`, can simulate individual traces for selected policies. For example, to run `bzip2` trace with Exclusive and NINE policies, follow the steps:

```
group21/$ ./driver (check usage and available flags)
group21/$ ./driver -trace bzip2.log_l1misstrace -parts 2 -simulate excl,nine
```

Refer to `README.md` for the code layout.

Results

Part 1

Trace	Policy	L2			L3			Hits in cache
		Lookups	Misses	Evictions (Back-invals)	Misses	Evictions	Invals	
bzip2	Incl	10657627	5398166	5389033 (941)	1446388	1413620	1413620	9211239
	Excl		5397576	5389384	889221	853441	5361796	9768406
	NINE		5397576	5389384	1445846	1413078	1413078	9211781
gcc	Incl	14610811	3036461	3018613 (9656)	1373402	1340634	1340634	13237409
	Excl		3029809	3021617	1242824	1202122	2989107	13367987
	NINE		3029809	3021617	1366248	1333480	1333480	13244563
gromacs	Incl	3431511	336851	328407 (252)	170531	137763	137763	3260980
	Excl		336724	328532	159302	118465	295887	3272209
	NINE		336724	328532	170459	137691	137691	3261052
h264ref	Incl	2348573	969678	955767 (5719)	342146	309378	309378	2006427
	Excl		965624	957432	143681	106041	927984	2204892
	NINE		965624	957432	333583	300815	300815	2014990
hmmer	Incl	3509765	1743421	1726529 (8700)	391226	358458	358458	3118539
	Excl		1735322	1727130	300046	261701	1696977	3209719
	NINE		1735322	1727130	376344	343576	343576	3133421
sphinx3	Incl	10753447	8820349	8793893 (18264)	8207362	8174594	8174594	2546085
	Excl		8815130	8806938	7220776	7179883	8774237	3532671
	NINE		8815130	8806938	8205144	8172376	8172376	2548303

Table 1: Number of lookups, misses, evictions, invalidations, and back-invalidations (when applicable) at L2 and L3 for traces with different cache policies.

Part 2

Trace	Accesses	Cold Misses	Capacity Misses		Conflict Misses		L3 misses (LRU)	
			LRU	Belady	LRU	Belady	Set-assoc	Fully-assoc
*bzip*2	5398166	119753	1241367	413616	85268	913019	1446388	1361120
gcc	3036461	773053	592969	165213	7380	435136	1373402	1366022
gromacs	336851	107962	61286	34530	1283	28039	170531	169248
*h*264*ref*	969678	63703	270140	44159	8303	234284	342146	333843
hmmer	1743421	75884	298304	77502	17038	237840	391226	374188
*sphinx*3	8820349	122069	8261943	2940477	-176650	5144816	8207362	8384012

Table 2: Miss classification for L3 misses using fully-associative LRU and Belady policies.

Analysis

The lookups at L2 do not take into account back-invalidation lookups in inclusive hierarchy. That number is separately provided in the result files as the field `Back-invalidation lookups`. The statistics conveyed in Table 1 implicitly are:

- Lookups in L3 is equal to number of misses at L2.
- Hits at L2 in all cases is the difference between lookups and misses at L2. Hits in L3 in all cases is the difference between lookups and misses at L3.
- Invalidations at L2 in inclusive hierarchy is equal to sum of evictions and back-invalidations at L2. Invalidations at L2 in exclusive and NINE hierarchy is equal to evictions at L2.
- Fills at L2 in all cases is equal to misses at L2. Fills at L3 in inclusive and NINE hierarchy is equal to misses at L3. Fills at L3 in exclusive hierarchy is equal to evictions at L2.

We make the following observations and inferences from Table 1:

- Miss rate at L2 is lower for exclusive and NINE policies as back-invalidations are absent. A back-invalidation may evict a hot block from L2 when it is victimized in L3. The overall hitrate, which is the ratio of accesses that hit in either level of cache (L2 or L3) to L2 lookups, is higher for non-inclusive cache policies, as the number of distinct blocks in the cache hierarchy are more.
- As the only difference between inclusive and NINE policies is the lack of back-invalidation in the latter, the relative improvement in overall hitrate correlates well with the increased proportion of back-invalidations. Let us take *bzip2* and *h264ref* for example. In *bzip2* trace, the percentage of back-invalidations among all invalidations at L2 is 0.017% compared to 0.595% for *h264ref.* The improvement in L2 hitrate for *bzip2* is 0.006% compared to 0.427% for *h264ref.* The coefficient of determination for the set of 6 traces is 0.83, where dependent variable is ratio of back-invalidations over total invalidations and independent variable is increase in cache hitrate from inclusive to NINE policy; this indicates a reasonable correlation.
- The exclusive cache policy performs better than both inclusive and NINE policies, as it increases the effective capacity of cache by disallowing repeated blocks at L2 and L3. The increase in overall hitrate achieved by NINE policy over inclusive policy is 0.17% on average while that achieved by exclusive policy over inclusive policy is 7.8% on average. The exclusive policy improves the performance over inclusive policy the most when the average reuse distance of blocks (number of unique blocks between two accesses to a given block) is higher than capacity of L3 (number of blocks for fully associative cache) but within capacity of L2 + L3. *sphinx3* enjoys a 27.9% increase in hitrate, owing to large working set (explained below).
- The invalidations at L3 in exclusive cache policy is the sum of L3 hits (when they are invalidated to maintain exclusivity) and L3 evictions (when a valid victim is invalidated). This is higher than the other two policies where L3 invalidatations occur only on evictions of valid blocks. Moreover, in all 6 traces, fills at L3 are higher for exclusive hierarchy, for which fills are equal to evictions at L2. As these blocks are filled into L3 via the cache interconnect, it increases traffic on the bus. Therefore, exclusive caches require higher bandwidth in the on-chip interconnects between levels of cache.

The implicit statistics conveyed in Table 2 are:

- The number of accesses in the simulated fully associative L3 cache is equal to the number of misses in L2 cache simulated in part1.
- Set associative LRU L3 misses are equal to cold misses + LRU capacity misses + LRU conflict misses.
- Cold misses for both replacement policies is the same since it refers to the number of distinct blocks accessed by the program (its working set) which is independent of the replacement policy.
- Set-associative LRU conflict misses is equal to the difference in set associative LRU misses and fully associative LRU misses.
- LRU capacity misses are more than Belady capacity misses as Belady, with a view of the future accesses, optimally evicts the block that is reused the furthest (or not reused at all), maximizing the cache capacity for an inclusive design.

For the subsequent discussion, the term reuse distance of a block is used to refer to the number of distinct blocks between two successive accesses to the same block. We gather the following inferences from Table 2:

- Cold misses represent the working set (distinct blocks accessed) of the application. The two traces *bzip2* and *sphinx3* have a similar working set but vastly different capacity misses in Belady implementation. Higher capacity misses in Belady for a similar working set indicates that more blocks on average have a reuse distance higher than the capacity of the cache since most blocks would have hit in the cache otherwise. This indicates that *sphinx3* accesses blocks with a reuse distance greater than the cache capacity. When the access pattern exceeds the cache capacity, LRU can thrash the cache which is happening in the case of *sphinx3* trace having a significantly high number of LRU capacity misses.
- The access pattern in *sphinx3* trace exceeds the cache capacity and leads to LRU thrashing the fully associative cache as argued in the point above. Since LRU can only evict a block from a set in the set associative LRU implementation, thrashing is limited to sets and the overall cache thrashing is the cumulative total of thrashing observed in the various sets. On the other hand, the entire cache is thrashed in a fully associative implementation which implies that LRU can be detrimental to performance in a fully associative cache as is observed in *sphinx3* trace. This also leads to negative conflict misses for this trace.
- The *gromacs* trace has a big working set reflected in the high number of cold misses. 32% of the accesses fall on distinct blocks which is significantly larger than the other traces. The capacity misses are quite low for both the Belady and LRU implementations. This indicates that the blocks which are reused fit in the cache and LRU is able to replace blocks. The trace still has a high miss rate of 50.24% in the fully associative L3 implementation owing to the big working set and hence the high cold misses. Many blocks are dead-on-arrival in this trace.
- *gromacs* has a larger working set (cold misses) than *h264ref* and the capacity misses in Belady are similar. This indicates that the effective working set (where blocks are reused) is within the cache level capacity. However, fully associative LRU exhibits a large difference in capacity misses. This indicates LRU is not able to evict dead-on-arrival blocks and must wait for DOAs to become least recently used, wasting cache capacity.
- *gcc* has, by far, the largest working set but its L3 cache hitrate is 54.77%, higher than *sphinx3* that has an L3 hitrate of 6.95%. 57% of blocks in *gcc* are dead-on-arrival (DOA) at L3 (that is, they are never accessed after the first access) while only 10% of the blocks in *sphinx3* are DOA. Moreover, a further 23.3% blocks in *gcc* are reused exactly once, while only 2.3% of blocks are reused exactly once in *sphinx3*. Essentially, even with a big working set, the *effective* working set where blocks are reused, is much smaller for *gcc*. In case of *sphinx3*, a lot of blocks are reused multiple times, with high average reuse distance, resulting in very low L3 hitrate.
- Since 57% of the blocks in *gcc* are dead-on-arrival, Belady would replace these blocks whenever there is a need for replacement whereas LRU would wait for such blocks to become least recently used before evicting them and possibly replace the useful blocks in the meanwhile which is happening in *gcc* leading to much higher capacity misses in LRU as compared to Belady.