COMPUTER SYSTEM ARCHITECTURE : TERM PROJECT REPORT

Development of Trace Driven Cache Memory Simulator

Hemant Kumar Pathak: CS20M007

Sathuri Bharath Kumar: CS20M011

May 24, 2021

1 PROBLEM STATEMENT

- To Design a trace driven simulator for Cache memory which takes trace file and configuration file as input
- To Perform three primary functions of cache (placement, locate and replacement)
- To Output statistics on cache reference, miss and hit.

2 EXISTING WORK

Trace-drivn cache simulation generally consisting of three stages-trace collection,trace reduction and trace processing[1,2].

TRACE COLLECTION

- Trace collection is the process of finding actual sequence of memory refrences which is normally consumes vey high volume of storage and processing time.

TRACE REDUCTION

- Trace reduction remove the redundant data which will result to reduce the trace size and processing time.

- Smith [2] proposed a “trace deletion technique”.He used an LRU-model of memory references and produced a reduced trace by deleting references that accessed the top D levels of the LRU stack of data.
- Puzak [2] proposed “trace stripping”This approach focuses on reducing traces for simulating set-associative caches. A direct-mapped cache (serving as a filter) is simulated and the miss references are recorded; they form a reduced trace.
- Apart form these many more methods like,Smith’s snapshot method records memory references at regular time intervals.
- A set selection method described by Puzak keeps only references that access some specific sets.
- Agarwal’s trace compaction combines Puzak’s reduction technique and Smith’s snapshot method.
- Laha proposed sampling based reducton this method can help when infinite number of traces are available so select portion of sample for experiment

TRACE PROCESSING

- Traces are fed to program that simulate the behavior of hypothetical memory system.
- Mattson proposed Stack based processing-all cache sizes under certain replacement policies with only a single pass through the trace file this approach work because they hold inclusion property-after any sequence of references, the contents of a cache are always a subset of any larger cache.
- Hill’s forest-simulation algorithm supports multiconfiguration simulation (fixing the cache associativity and varying the number of sets). Another algorithm studied by Hill is all-associativity simulation,which enables both the number of sets and the associativity to be varied with just slightly more overhead than forest simulation.

Conclusion: all the above methods have their own importance and are useful in specific purposes

3 OUR APPROACH

The Approach that we have followed in implementing this simulator is discussed here in detail.

NOVELTY

- For trace processing we have introduced the concept of unordered set which uses the hashing so search time is O(1).
- We have introduced the unique mapping module which can be able to perform for all mapping techniques according to parameter passed.

STEPS

1. Writing the trace file in the given format which will be given as input to the simulator
2. Writing the program to simulate all the cache primary functions
   a) Implementing Cache Placement
   b) To locate
   c) Replacement functions
3. Getting statistics on various parameters as output from the simulator from the trace input and configuration file.

4 EXPERIMENTAL SETUP

For our simulator, we have taken 3 configuration files for 3 cache sizes of 2MB, 1MB, and 512KB respectivly. Also we have 5 trace files which will be an input for our simulator. (gcc.trace, gzip.trace, mcf.trace, swim.trace, twolf.trace). The program will take these two files(configuration and trace files) as input and produces output files respectively based on the parameters.

4.1 MAPPING TECHNIQUES

OUR IDEA We thought its better to implement one set associative function with k as input or parameter based on value we call these functions as all are variants of set associative mapping So, We have written a common function called mapping which is called from each type of mapping technique which would function based on k value from trace input file.

- 1-way set associative mapping is direct mapping (1 block in 1 set).
- N-way set associative mapping is fully associative mapping(n blocks in 1 set).
- K-way set associative mapping is default set associative mapping(k-blocks in 1 set).

IMPLEMENTATION

- DIRECT by cache block = mod of main block (this is nothing but 1 set associative where each block will be in 1 set)
- FULLY ASSOCIATIVE can be placed anywhere as whole cache is one set(this is nothing but n set associative where all blocks in cache are in 1 set)
- SET ASSOCIATIVE cache set = mod of main block

4.2 CACHE REPLACEMENT

- LRU by checking for least recently used or the previous requests
- FIFO by replacing from beginning of cache vector(old ones first replaced)

4.3 HIT AND MISS RATIO

CALCULATING HIT AND MISS RATIO FROM ABOVE by taking cache into a set and checking in O(1) for the present element. if its present already, then its a hit or else its a miss. So accordingly, we increment the count of hit and miss for all the addresses in the trace file according to the parameters in the configuration file.

$$hitrate = \frac{numberofhits}{numberofmiss + numberofhits} \tag{4.1}$$

i.e., total hits divided by total number of requests.

$$hit\% = hitrate * 100 \tag{4.2}$$

i.e., total hits * 100 gives hit percentage.

$$missrate = \frac{numberofmiss}{numberofmiss + numberofhits} \tag{4.3}$$

i.e., total miss divided by total number of requests.

$$miss\% = missrate * 100 \tag{4.4}$$

i.e., total miss * 100 gives miss percentage.

5 RESULTS AND ANALYSIS

5.1 RESULTS TABLE

Trace_file	Config_file		Direct_map				K-way set(k=2)				K-way set(k=4)				fully_associative			
			LRU_Replacement		FIFO_Replacement		LRU_Replacement		FIFO_Replacement		LRU_Replacement		FIFO_Replacement		LRU_Replacement		FIFO_Replacement	
			HIT	MISS	HIT	MISS	HIT	MISS	HIT	MISS	HIT	MISS	HIT	MISS	HIT	MISS	HIT	MISS
gcc.trc	cache_configure_2mb.cfg	hit/miss percentage(%)	35.93	64.07	33.02	66.98	51.45	48.54	50.63	49.37	60.76	39.34	62.47	37.53	74.49	25.51	83.4	16.6
	cache_configure_1mb.cfg	hit/miss percentage(%)	35.7	64.3	33.02	66.98	51.48	48.52	50.63	49.37	60.94	39.06	62.61	37.39	73.91	26.09	79.49	20.51
	cache_configure_512kb.cfg	hit/miss percentage(%)	32.89	67.11	33.02	66.98	50.46	49.54	50.63	49.37	62.42	37.58	62.81	37.19	72.75	27.25	73.28	26.72
			HIT	MISS	HIT	MISS	HIT	MISS	HIT	MISS	HIT	MISS	HIT	MISS	HIT	MISS	HIT	MISS
gzip.trc	cache_configure_2mb.cfg	hit/miss percentage(%)	61.76	38.34	61.69	38.31	64.32	35.67	64.3	35.7	65.48	34.52	66.51	34.49	66.47	33.53	66.51	34.49
	cache_configure_1mb.cfg	hit/miss percentage(%)	68.37	31.63	61.7	38.3	64.33	35.67	64.31	35.69	65.49	34.51	65.5	34.5	66.33	33.67	66.36	33.64
	cache_configure_512kb.cfg	hit/miss percentage(%)	61.72	38.28	61.69	38.31	64.3	35.7	64.3	35.7	60.87	39.13	65.49	34.51	66.05	33.95	66.07	33.93
			HIT	MISS	HIT	MISS	HIT	MISS	HIT	MISS	HIT	MISS	HIT	MISS	HIT	MISS	HIT	MISS
mcf.trc	cache_configure_2mb.cfg	hit/miss percentage(%)	0.38	99.62	0.43	99.57	0.54	99.46	0.64	99.36	0.65	99.35	0.77	99.23	0.81	99.19	0.99	99.01
	cache_configure_1mb.cfg	hit/miss percentage(%)	0.41	99.59	0.44	99.56	0.54	99.46	0.64	99.36	0.66	99.34	0.78	99.22	0.8	99.2	0.97	99.03
	cache_configure_512kb.cfg	hit/miss percentage(%)	0.42	99.58	0.44	99.56	0.61	99.39	0.64	99.36	0.74	99.26	0.78	99.22	0.85	99.15	0.89	99.11
			HIT	MISS	HIT	MISS	HIT	MISS	HIT	MISS	HIT	MISS	HIT	MISS	HIT	MISS	HIT	MISS
swim.trc	cache_configure_2mb.cfg	hit/miss percentage(%)	18.47	81.53	18.51	81.49	60.01	39.99	54.7	45.3	74.03	25.97	76.46	26.54	84.69	15.31	87.09	12.91
	cache_configure_1mb.cfg	hit/miss percentage(%)	18.4	81.6	18.47	81.53	60.07	39.93	54.7	45.3	73.82	26.18	73.47	26.53	82.3	17.7	84.45	15.55
	cache_configure_512kb.cfg	hit/miss percentage(%)	17.45	82.55	18.45	81.55	56.57	43.53	54.68	45.32	74.05	25.95	73.53	26.47	80.08	19.92	80.92	19.08
			HIT	MISS	HIT	MISS	HIT	MISS	HIT	MISS	HIT	MISS	HIT	MISS	HIT	MISS	HIT	MISS
twolf.trc	cache_configure_2mb.cfg	hit/miss percentage(%)	36.69	60.31	39.25	60.75	60.07	39.93	59.58	40.42	76.99	23.11	76.36	23.63	97	3	96.89	3.11
	cache_configure_1mb.cfg	hit/miss percentage(%)	39.63	60.37	39.21	60.79	60.08	39.92	59.59	40.41	77.01	22.99	76.36	23.64	95.23	4.77	95.05	4.95
	cache_configure_512kb.cfg	hit/miss percentage(%)	39.28	60.72	39.23	60.77	59.68	40.31	59.59	40.41	76.47	23.53	76.41	23.59	87.35	12.65	86.79	13.21

5.2 Plot of GCC Trace File

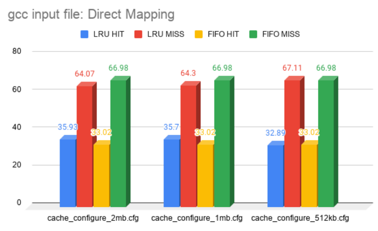

Figure 5.1: gcc: Direct Mapping

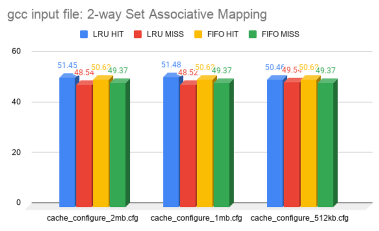

Figure 5.2: gcc: 2-way Set Associative Mapping

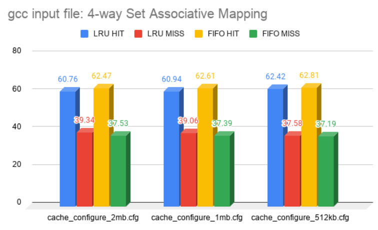

Figure 5.3: gcc: 4-way Set Associative Mapping

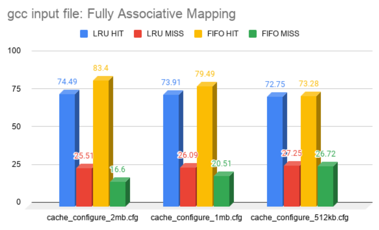

Figure 5.4: gcc: Fully Associative Mapping

The Above figures 5.1, 5.2, 5.3, 5.4 are plots of the table data related to gcc input trace file for all the 4 mappings. (Direct Mapping, 2-way and 4-way Set Associative Mapping, Fully set Associative Mapping)

From the above plots, We can infer that, for LRU, Hit rate increases with increase in Cache size in Direct Mapping and for FIFO, Hit rate remains same.

In 2-way Set Mapping, for LRU replacement, Hit rate increases first from 512 kb cache to 1MB cache where as it decreases slightly from 1MB cache to 2MB cache and for FIFO, hit remains constant.

In 4-way Set Mapping, for LRU replacement, Hit rate decreases with increase in cache size and for FIFO, hit decreases slightly with increase in size.

In Fully Set Associative Mapping, for LRU replacement, Hit rate increases with increase in cache size and also for FIFO, hit rate increases with increase in cache size.

6 References

Github Repo: `https://github.com/Bharathbrothers/CSA_TERM_PROJECT`

1.ACM Computing Surveys, Vol. 29, No. 2, June 1997

2.ACM Transactions on Computer Systems, Vol 9, No, 3, August 1991.