CS6600 Assignment 2: Cache Replacement Policies

Variance Prediction of Mockingjay Reuse Distance Predictor

Arun Krishna AMS(EE19B001), Lakshya J(EE19B035)

1. Abstract:

MockingJay cache replacement policy directly mimics Belady's MIN Policy in a simple and effective way. Mockingjay uses a PC Based Predictor to learn each cache line's reuse distance and evicts lines based on their predicted time of reuse. Here, we explore multiple methods to determine the accuracy of/variance in the reuse distance predictor - and inject them in the eviction algorithm. The Eviction algorithm is modified to have a higher tolerance for RDP's of PCs having higher variance/lower accuracy. When conflicted between choosing the victim cache line (because of lower accuracy), the eviction policy chooses the cache line that has been least recently used.

2. Introduction to Mockingjay

MockingJay determines at each cache access, the line's estimated time of arrival as the sum of the current time stamp and the line's predicted reuse distance. To predict the reuse distance, a PC Based predictor is used to estimate cache line's reuse distance.

A sampled cache is used to maintain a long history of past cache accesses (8 x #ways of associativity). This is used to track the past reuse distances to train and predict the RDP in the future. Each entry of the sampled cache has the cacheline's last time stamp and their last PC Signature. The RDP is indexed by the PC. As lines are reused in the sampled cache, the RDP entry corresponding to the PC that last accessed the cache line is updated.

The cache maintains an ETR Estimated Time Remaining counter for each cache line. Upon insertion/access of a cacheline, the ETR counter is initialised with the cacheline's RDP. For every access in the set the ETR counters of the set decrements by 1. During eviction, the cache line with the highest ETR value is evicted.

3. Variance Prediction Based on ETR

If an ideal RDP with 100% accuracy in predicting the reuse distance is used in the replacement policy, every cache line will be accessed when the ETR counter reaches exactly zero. But if the RDP is underestimated or overestimated the ETR counter at the time of access will be non zero. Thus the value of the ETR at the time of access can be used to estimate the degree of accuracy/variance in the reuse prediction.

3.1. Version 1: Discrete Implementation:

In Version 1, We estimate the degree of variance in the reuse prediction with respect to the initial value of ETR of the cacheline that is being accessed. In MockingJay when a cacheline is being accessed, the ETR value of the cacheline is initialized with RDP of the cacheline. Here, we also store this value in etr_initial registers. Unlike ETR counters, etr_initial doesn't decrement for every access in the set. The PC Signature that accessed the cacheline is also stored in etr_signature.

At the time of cache hit, we determine the variance of the `RDP` of the PC that last accessed the cacheline into 8 levels.

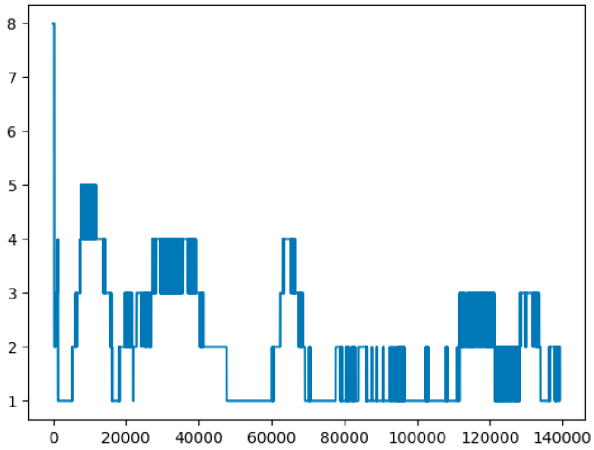
```
LEVEL 'i' IF absolute(etr[way]) LIES BETWEEN ((i-1)*etr_initial[way]/8),
                                        AND  (i*etr_initial[way]/8) )
```

`level` 0 indicates that the `RDP` is quite accurate, while `level` 7 indicates that `RDP` has a very high variance. The accuracy is averaged out for the last 8 cache hits for that `PC` and is stored along with the `RDP` as `rdp_accuracy`.

```
IF cache hit THEN:
     IF absolute(etr[way]) < absolute(etr_initial[way])/8 THEN
         SET rdp_accuracy_level <-- 0
     ELSE IF absolute(etr[way]) < 2*absolute(etr_initial[way])/8 THEN
         SET rdp_accuracy_level <-- 1
     ELSE IF absolute(etr[way]) < 3*absolute(etr_initial[way])/8 THEN
         SET rdp_accuracy_level <-- 2
     ELSE IF absolute(etr[way]) < 4*absolute(etr_initial[way])/8 THEN
         SET rdp_accuracy_level <-- 3
     ELSE IF absolute(etr[way]) < 5*absolute(etr_initial[way])/8 THEN
         SET rdp_accuracy_level <-- 4
     ELSE IF absolute(etr[way]) < 6*absolute(etr_initial[way])/8 THEN
         SET rdp_accuracy_level <-- 5
     ELSE IF absolute(etr[way]) < 7*absolute(etr_initial[way])/8 THEN
         SET rdp_accuracy_level <-- 6
     ELSE THEN
         SET rdp_accuracy_level <-- 7

     SET rdp_accuracy(etr_signature[set][way]) <-- rdp_accuracy_level

     SET etr[way] <-- rdp(current_pc_signature)
     SET etr_initial[way]  <-- etr[way]
     SET etr_signature[way]  <-- current_pc_signature
```



Above accuracy levels were obtained when MockingJay replacement policy is used for one of the pc_signatures of lbm_94B trace - Graph indicates the degree of accuracy of RDP at different instances. Initially starts with Level 7 and reduces over period of time globally with occasional jumps to level 3

At the time of finding a cacheline to be evicted, the variance level is found for the cacheline by indexing the `rdp_accuracy` using the `PC` signature of the `PC` that last accessed the cacheline(Obtained from `etr_signature`). The variance in the cacheline is then determined using `etr_initial` * `level` / 8.

```
DEFINE FUNCTION varianceOf(set, way):
    SET rdp_accuracy_level <-- rdp_accuracy(etr_signature[set][way])
    RETURN variation = etr_initial[set][way] * rdp_accuracy_level  / 8;
ENDFUNCTION
```

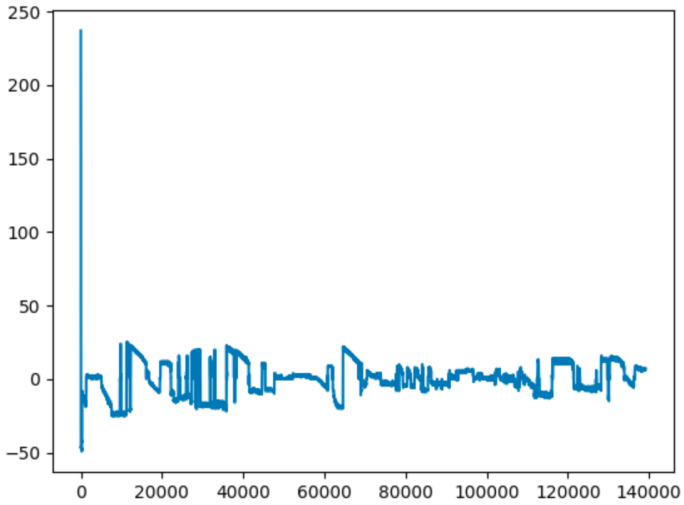
3.2. Version 2: Continuous Implementation:

In version 2, we estimate the variance as the absolute value of the `ETR` itself at the time of access.

```
IF cache hit THEN:

      SET rdp_accuracy(etr_signature[set][way]) <-- etr[way]

      SET etr[way] <-- rdp(current_pc_signature)
      SET etr_initial[way]  <-- etr[way]
      SET etr_signature[way]  <-- current_pc_signature
```



Above figure were obtained when MockingJay replacement policy is used for one of the pc_signatures of lbm_94B trace - Indicates the estimated variation of RDP at different instances. Initially starts with very high variation and stabilizes globally between -30 and 30

At the time of finding a cacheline to be evicted, the variance is found for the cacheline by indexing the `rdp_accuracy` using the `PC` signature of the `PC` that last accessed the cacheline(Obtained from `etr_signature`).

```
DEFINE FUNCTION varianceOf(set, way):
      SET rdp_accuracy <-- rdp_accuracy(etr_signature[set][way])
      RETURN variation = abs(rdp_accuracy);
  ENDFUNCTION
```

3.3. Version 5: Sampled Cache Set ETR and Continuous Implementation:

We try to improve the variance predictions and its influence in eviction policy from version 1 and 2 by

Version 1 and 2 utilizes `ETR` of the cachelines available in the `cache`. In version 5, We assign one `ETR` counter for each cacheline in the `sampled_cache`. The `sampled_cache` has information on 256 lastly accessed unique cachelines of the set compared to the 16 unique cachelines in the `cache`. Since the amount of data available is increased, we expect better performance.

Version 1 and 2 determine the variation at the time of eviction. Suppose a cacheline is accessed at time '`t1`'. Since `RDP` accuracy is continuously updated, At the time of eviction (t2 >> `t1`), the `RDP` accuracy could be very different from the time the cacheline was previously accessed(`t1`). Thus variation error is determined at the time of access and is stored in `etr_error` for each cacheline in the cache.

```
DEFINE FUNCTION varianceOf(set, way):
      RETURN variation = abs(etr_error[set][way]);
  ENDFUNCTION
```

`etr_error` is updated during the cacheline access with the `RDP` accuracy predicted by the `PC` along with the `RDP`.

```
          SET etr[set][way] <-- rdp(current_pc_signature)
          SET etr_error[set][way] <-- rdp_accuracy(current_pc_signature)
```

3.4. Version 3 & 4:
Variance Prediction of RDP Based on Difference in Elapsed time and RDP:

Versions 3 and 4 are very similar to versions 1 and 2. Unlike versions 1 and 2 which uses `ETR` at the time of access - Versions 3 and 4 estimates variation in `RDP` as a difference between the predicted `RDP` and the elapsed time.

```
IF hit IN sampled_cache:
  SET elapsed_time <-- current_timestamp - last_access_timestamp
  SET rdp_accuracy(last_access_pcsignature) <-- elasped_time - rdp(last_access_pcsignature)
```

4. Eviction Policy

4.1 Why should variation be included in Eviction Policy:

Mockingjay evicts cachelines that have the longest `ETR - Estimated Time Remaining`.
Consider the following example:

At a certain instant where the LLC is forced to evict cacheline from `Set 1620`, following `ETA` and `RDP` variation were observed:

```
Set:    2038
ETA:
44(-5)   1(0)   15(0)   31(0)   6(-3)   11(0)   43(-1)  14(6)   8(5)    18(6)   28(6)   44(0)
4(6)    17(-3)  27(-3)  5(0)
```

Values within () are the variation in RDP predicted based on Version 2

One can observe that the '`way 0`' has the highest `ETR` with `44` with a variation of `5`, while '`way 6`' has an `ETR` of `43` with a predicted variation of `1`. While mockingjay evicts cacheline in '`way 0`', '`way 6`' is also be a good choice to evict. Why should '`way 0`' be preferred even when the accuracy of predicted RDP is less compared to '`way 6`'?

Hence variation is also used as a factor during eviction.

4.1 Eviction Policy:

During eviction, the cacheline with the maximum `ETR` value is chosen as `max_etr_way`. The degree of accuracy/variance in the `RDP` of the cacheline `max_etr_way` is found. All cachelines whose `abs(etr)` lie within this tolerance i.e., "`abs(max_etr) +/- abs(variance)`" is a possible eviction candidate. Tie among the eviction candidates is chosen based on `Least Recently Used Policy`.

```
SET max_etr <--- 0
SET max_etr_way <--- 0

FOR (way = 0 UNTIL way = LLC_WAYS - 1)
    IF absolute(etr[way]) > max_etr THEN
        SET max_etr <--- absolute(etr[way])
        SET max_etr_way <--- way


SET variance <-- varianceOf(set, way)
SET ways_within_variance <-- {}
FOR (way = 0 UNTIL way = LLC_WAYS - 1)
    IF absolute( absolute(etr[max_etr_way]) - absolute(etr[way]) ) < variance THEN
         ADD way TO ways_within_variance

AMONG ALL way IN ways_within_variance FIND least_recently_used_way
eviction_way <-- least_recently_used_way
```

5. Results

TRACE	Mockingjay	SHIP	LRU	Hawkeye	Version 1	Version 2	Version 3	Version 4	Version 5
astar	0.41787	0.42403	0.42450	0.42037	0.42466	0.42469	0.42467	0.42468	0.42468
bwave	1.63476	1.63646	1.6365	1.63803	1.63512	1.63537	1.63529	1.63539	1.63529
bzip	1.34836	1.32075	1.30547	1.3441	1.31384	1.31401	1.32155	1.30801	1.31274
cactus	1.3055	0.92347	0.92182	1.21799	1.04185	0.95041	1.23995	1.20387	1.12434
calculix	2.63578	2.68247	2.66895	2.53231	2.65375	2.66455	2.63372	2.65538	2.6628
gcc	0.48970	0.47844	0.47721	0.48613	0.47445	0.47545	0.47515	0.47596	0.47600
gems	0.31995	0.29586	0.29575	0.3191	0.31121	0.30066	0.31892	0.31097	0.31021
lbm	0.35069	0.25224	0.25241	0.25129	0.27145	0.25025	0.31035	0.26764	0.27101
leslie	0.49927	0.45434	0.44468	0.47749	0.48025	0.45224	0.50488	0.45423	0.45229
libquant	0.49781	0.47123	0.47133	0.49478	0.46621	0.46554	0.47045	0.46579	0.46607

Unfortunately none of the versions on overall performed better than all the other replacement policies

- **'A star' trace: Performed better than all the existing policies**
- **'bwave' trace:** Results were very similar for all replacement policies
- **'bzip' trace:** Performed better than LRU but worse than all others
- **'cactus' trace:** Version 3 performed better than Hawkeye, SHIP and LRU but worse than Mockingjay
- **'calculix' trace:** Performed better than Mockingjay & Hawkeye but worse than SHIP and LRU
- **'gcc' trace: Performed worse than all**
- **'Gems' trace:** Version 1 performed as same as Mockingjay and Hawkeye & better than others
- **'lbm' trace:** Version 1,3,4,5 performed better than everyone except Mockingjay.
- **'leslie' trace: Version 3 performed better than all existing policies**. Other versions performed better than LRU & SHIP, but not Hawkeye(except Version 1)
- **'libquant' trace: Performed worse than all.**

6. Other Ideas and Failed Experiments

- `LRU` performed better for `Writeback` accesses when compared to other replacement policies, but worse for `ROF` & `Load` accesses. We tried to understand the reason behind this and exploit to get higher performance - but no leads were made.
- Different `PC Signatures` were exploited
    - Including previous PCs in the PC signature
    - Including/Not Including Hit/Miss information in the signature
    - Including/Not Including Type of access: `ROF`, `Writeback`, `Load` instruction type in the signature

  No measurable performance improvement was found
- Different `RDP Update` policies were explored. No measurable performance improvement was found.