CS204
Computer Architecture

Implementing Utility Based Cache Partitioning

Submitted to:

Dr. Shirshendu Das

Submitted by:

Shyam Prajapati (2020CSB1110)
Arya Phalke (2020CSB1107)
Ashish Sharma (2019MCB1213)
Sushil Kumar (2020CSB1132)
Armaan Sharma (2020CSB1039)
Vinit Hagone (2020CSB1361)

Utility Cache Partitioning:

- UCP divides the cache among the cores based on the algorithm which decides the division of ways based on utility.

- It has utility monitoring circuits (UMON) which have an auxiliary tag directory (ATD) which keeps count of hit rates.

- Based on the various values of UMON for different possible combinations of way partitions it chooses the optimum partition where in hit rate is maximum or miss rate is minimum.

- ATD is also just like a cache which uses LRU policy to calculate the division of cache partitioning.

- Each ATD contains blocks of their corresponding cores & possible partitions are applied based on same.

- The number of possible cache partitions increases exponentially and maintaining tag directories and hit counters for each may cause overhead.

- UMON Global which applies uniform partitioning on all sets is possible option.

- Here we have implemented Dynamic Set Sampling (DSS) for UMON which takes sampled sets (64) and applies the partitioning on them and then the result is applied to the follower sets.

- Taking only 64 sets for sampling reduces the overhead and also gives the required result.

Implementation:

New files created or modified:

- *ucp.llc_repl*
- *llc_find_victim*
- *atd.h*
- *cache.cc*

The base replacement policy is LRU which has been further modified to implement the UCP algorithm.

1. Created a new file in the replacement folder: ucp.llc_repl

The file has following functions:

- *llc_ find_victim*
- *find_victim*
- *llc_update_replacement_state*
- *llc_initialize_replacement*

2. Created a new include file to implement ATD: atd.h

The file has following properties:

- It contains two classes namely *class blockatd & class ATD.*
- class blockatd contains valid bit, tag bit, lru bit.
- class ATD contains 2D array of blockatd & an array to maintain the umon global counter.

3. Made changes in cache.cc to implement the UCP policy:

The following functions were added:

- *void atd_insert*
- *int atdupdate*
- *void atdfill*
- *void partition*

Created two new atds for implementing dual core partition. The above-mentioned functions were further used to apply the UCP partition.

atd_insert:

- It inserts new block in the atd and checks if there is any available empty space in atd.
- If no empty space, then the it evicts the block which has maximum lru and then the lru value of new inserted block is 0.

atdupdate:

- It checks on which block there is a hit that is tells the block and way of the hit block.
- It then updates the lru of that block to 0 and increments the lru of remaining blocks' lru values by 1.
- Also returns the hit value which is the block number.

atdfill:

- It checks whether the block coming into the atd is a hit or miss.
- If it's a hit then the umonglobal is updated & lru is incremented.
- Fills the ATD according to the hits and misses encountered.

void partition:

- This algorithm tries out all possible combinations of the 16 ways and returns the best optimal partition.
- It tells the optimal one by checking the utilities of the combinations.
- As per the UCP research paper the same one is implemented.

Sampled Sets:

- Each 64^{th} set is used as a sampled set. Calculated by taking modulus of set number with 64 and equating the same to zero.
- ATD takes these sets and finds the optimal partition using UCP and then the result is applied on follower sets.
- With every 2.5 mil cycles the partition changes as per the output of our policy.

How the partition changes?

- After every 2.5 million cycles the partitioning algorithm is called & dynamically it gets updated.
- The umon counters for each ATD are halved after every call of partition.
- We have implemented our code for 2 core partition.
- We have implemented the ATD and even done the same for static partition.

How to run the code?

For N core partition run_Ncore.sh is a file which needs to be created to be able to run the code.

We have attached the files for 2 cores and 4 cores.

Also, when we are running LRU we have to first comment out the lines where in ucp variables are declared.

Following commands are for 2 core partition:

- **Build:**

UCP:
./build_champsim.sh bimodal no no no no ucp 2

LRU:
./build_champsim.sh bimodal no no no no lru 2

- **Run:**

UCP:
./run_2core.sh bimodal-no-no-no-no-ucp-2core 1 10 trace_core0_name.trace.xz trace_core1_name.trace.xz

LRU:
./run_2core.sh bimodal-no-no-no-no-lru-2core 1 10 trace_core0_name.trace.xz trace_core1_name.trace.xz

Results:

UCP:

For trace : leslie3d_1186B

```
Region of Interest Statistics

CPU 0 cumulative IPC: 0.612907 instructions: 10000001 cycles: 16315681
L1D TOTAL     ACCESS:    3142631  HIT:    2831459  MISS:     311172
L1D LOAD      ACCESS:    2068185  HIT:    1803923  MISS:     264262
L1D RFO       ACCESS:    1074446  HIT:    1027536  MISS:      46910
L1D PREFETCH  ACCESS:          0  HIT:          0  MISS:          0
L1D WRITEBACK ACCESS:          0  HIT:          0  MISS:          0
L1D PREFETCH  REQUESTED:          0  ISSUED:          0  USEFUL:
```

For trace : bzip2_281B

```
CPU 1 cumulative IPC: 0.780773 instructions: 10000000 cycles: 12807821
L1D TOTAL     ACCESS:    2259259  HIT:    2081328  MISS:     177931
L1D LOAD      ACCESS:    1630117  HIT:    1478526  MISS:     151591
L1D RFO       ACCESS:     629142  HIT:     602802  MISS:      26340
L1D PREFETCH  ACCESS:          0  HIT:          0  MISS:          0
L1D WRITEBACK ACCESS:          0  HIT:          0  MISS:          0
L1D PREFETCH  REQUESTED:          0  ISSUED:          0  USEFUL:
L1D AVERAGE MISS LATENCY: 183.963 cycles
```

LRU:

For trace: leslie3d_1186B.trace.xz

```
Region of Interest Statistics

CPU 0 cumulative IPC: 0.542285 instructions: 10000001 cycles: 18440482
L1D TOTAL     ACCESS:    3119049  HIT:    2807874  MISS:     311175
L1D LOAD      ACCESS:    2054698  HIT:    1790432  MISS:     264266
L1D RFO       ACCESS:    1064351  HIT:    1017442  MISS:      46909
L1D PREFETCH  ACCESS:          0  HIT:          0  MISS:          0
L1D WRITEBACK ACCESS:          0  HIT:          0  MISS:          0
L1D PREFETCH  REQUESTED:       0  ISSUED:       0  USEFUL:
L1D AVERAGE MISS LATENCY: 172.095 cycles
```

For trace: bzip2_281B.trace.xz

```
CPU 1 cumulative IPC: 0.799724 instructions: 10000000 cycles: 12504322
L1D TOTAL     ACCESS:    2261923  HIT:    2083998  MISS:     177925
L1D LOAD      ACCESS:    1632611  HIT:    1481021  MISS:     151590
L1D RFO       ACCESS:     629312  HIT:     602977  MISS:      26335
L1D PREFETCH  ACCESS:          0  HIT:          0  MISS:          0
L1D WRITEBACK ACCESS:          0  HIT:          0  MISS:          0
L1D PREFETCH  REQUESTED:       0  ISSUED:       0  USEFUL:       0  USELE
L1D AVERAGE MISS LATENCY: 197.149 cycles
L1I TOTAL     ACCESS:    1760499  HIT:    1760487  MISS:         12
```