# CPEN411 Assignment2 Report

Shidi Xi 90506643

October 27, 2023

## 1 LIP

During a cache hit, LIP behaves the same way as LRU where the `lru_update()` function is called in the `CACHE::llc_update_replacement_state()` function. During a cache miss, the LRU line is evicted, but new line will remain as LRU. Essentially, we do not call the `lru_update()` function.

```
uint32_t CACHE::llc_find_victim()
{
    // baseline LRU
    return lru_victim();
}

// called on every cache hit and cache fill
void CACHE::llc_update_replacement_state(set, way)
{
    if (hit && (type == WRITEBACK)) // writeback hit does not update LRU state
        return;
    else if (hit) {
        return lru_update(set, way);
    }
    else if (!hit) {
        // The newly filled cache line will remain LRU
        // Other lines's LRU will remain the same
        return;
    }
}
```

## 2 BIP

During a cache hit, BIP behaves the same way as LRU where the `lru_update()` function is called in the `CACHE::llc_update_replacement_state()` function. During a cache miss, the LRU line is evicted, but new line will remain as LRU with a probability of 95%. The new line will be marked as MRU with a probability of 5%.

```
    void CACHE::llc_update_replacement_state()
{
    int random_num;

    if (hit && (type == WRITEBACK)) // writeback hit does not update LRU state
        return;
    else if (hit) {
        return lru_update(set, way);
    }
    else if (!hit) {
        // The newly filled cache line will remain LRU with a probability of 95%
        // Other lines's LRU will remain the same
        std::random_device rd;
        std::uniform_int_distribution<int> dis(0, 99);
        random_num = dis(rd);
        if (random_num < 5) {
            return lru_update(set, way);
        }
        else {
            return;
        }
    }
}
```

# 3 DIP

DIP is more complicated to implement. First, in the `llc_initialize_replacement()` function, we random choose some sets within the cache to be the leader sets. The indices of the leader sets are stored in the array `rand_sets`. The PSEL is also initialized to be 0 in this function. During a cache hit, we will do normal RLU. During a cache miss, the LRU line is evicted. The new line will either be labeled as the MRU, or it will follow BIP. The `is_it_leader()` function is first called to determine whether the new line is in the follower set or the leader set. If it is a leader, then it will follow the respective insertion policy, and PSEL will be updated. If it is a follower, then it will follow one of the policies, according to the most significant bit of PSEL.

```
#define NUM_POLICY 2
#define SDM_SIZE 64
#define TOTAL_SDM_SETS NUM_CPUS*NUM_POLICY*SDM_SIZE
#define PSEL_WIDTH 10
#define PSEL_MAX 1023

uint32_t PSEL[NUM_CPUS];
unsigned rand_sets[TOTAL_SDM_SETS];

// initialize replacement state
void CACHE::llc_initialize_replacement()
{
    srand(time(NULL));
    unsigned long rand_seed = 1;
    unsigned long max_rand = 1048576;
    uint32_t my_set = LLC_SET;
    int do_again = 0;
    for (int i=0; i<TOTAL_SDM_SETS; i++) {
        do {
            do_again = 0;
            rand_seed = rand_seed * 1103515245 + 12345;
            rand_sets[i] = ((unsigned) ((rand_seed/65536) % max_rand)) % my_set;
            printf("Assign rand_sets[%d]: %u  LLC: %u\n", i, rand_sets[i], my_set);
            for (int j=0; j<i; j++) {
                if (rand_sets[i] == rand_sets[j]) {
                    do_again = 1;
                    break;
                }
            }
        } while (do_again);
        printf("rand_sets[%d]: %d\n", i, rand_sets[i]);
    }

    for (int i=0; i<NUM_CPUS; i++)
        PSEL[i] = 0;
}

int is_it_leader(uint32_t cpu, uint32_t set)
{
    uint32_t start = cpu * NUM_POLICY * SDM_SIZE,
             end = start + NUM_POLICY * SDM_SIZE;

    for (uint32_t i=start; i<end; i++)
        if (rand_sets[i] == set)
            return ((i - start) / SDM_SIZE);

    return -1;
}

int generate_randnum(void)
{
    int random_num;

    std::random_device rd;
    std::uniform_int_distribution<int> dis(0, 99);
    random_num = dis(rd);

    return random_num;
}

// find replacement victim
uint32_t CACHE::llc_find_victim()
{
    // baseline LRU
    return lru_victim(cpu, instr_id, set, current_set, ip, full_addr, type);
}

// called on every cache hit and cache fill
void CACHE::llc_update_replacement_state()
{
    int random_num;

    if (hit && (type == WRITEBACK)) // writeback hit does not update LRU state
        return;
    else if (hit) {
        return lru_update(set, way);
    }
    else if (!hit) {
        int leader = is_it_leader(cpu, set);

        if (leader == -1) { // the set is a follower
        // 10-bit PSEL, when MSB = 1 and others 0, it is ’d512, follow BIP
```

```
                if (PSEL[cpu] >= 512) {
                    random_num = generate_randnum();
                    if (random_num < 5) {
                        return lru_update(set, way);
                    }
                    else {
                        return;
                    }
                }
                else if (PSEL[cpu] <= 511) { // follow LRU
                    return lru_update(set, way);
                }
            }
            else if (leader == 0) { // leader 0: BIP
                if (PSEL[cpu] > 0) {
                    PSEL[cpu] --;
                }
                random_num = generate_randnum();
                if (random_num < 5) {
                    return lru_update(set, way);
                }
                else {
                    return;
                }
            }
            else if (leader == 1) { // leader 1: LRU
                if (PSEL[cpu] < PSEL_MAX) {
                    PSEL[cpu] ++;
                }
                return lru_update(set, way);
            }
        }
    }
```

## 4 Pseudo Tree LRU

This policy is implemented with reference to this tutorial: https://www.cs.virginia.edu/luther/3330/S2022/tree-plru.html#fnref1. We first initialize the binary pseudo RLU tree data structure in the llc_initialize_replacement() function, where it is stored in a 2D array. During a cache miss, a victim is found by walking down the tree according to the node bits in the llc_find_victim() function. Then the node bits are updated by up-tracing the tree in llc_update_replacement_state(). During a cache hit, the node bits are updated in the same way.

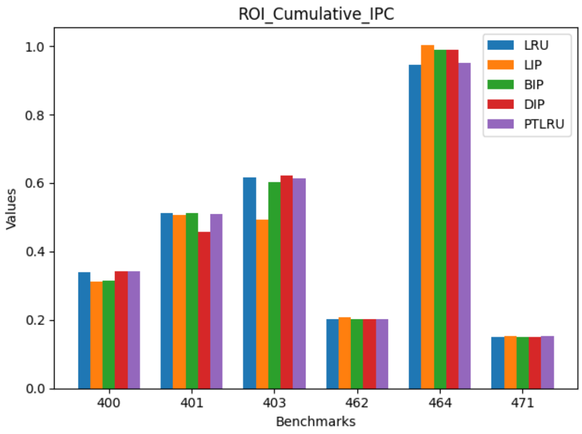
```
#define TREE_SIZE  LLC_WAY-1
#define TREE_DEPTH (int)log2(LLC_WAY)
bool plru_tree[LLC_SET][TREE_SIZE];

// initialize replacement state
void CACHE::llc_initialize_replacement()
{
    // Initialize PLRU trees
    for(int i=0; i<LLC_SET; i++) {
        for(int j=0; j<TREE_SIZE; j++) {
            plru_tree[i][j] = 0;  // Initially, all bits are set to 0.
        }
    }
}

// find replacement victim
uint32_t CACHE::llc_find_victim()
{
    // Forward pass of the tree following the arrows
    uint32_t lru_way = 0;
    int node = 0;
    int i = 0;
    int lru_node[TREE_DEPTH];

    while (node<TREE_SIZE) {
        lru_node[i] = plru_tree[set][node];
        node = 2*node+1+lru_node[i];
        i++;
    }

    for (i=0; i<TREE_DEPTH; i++) {
        lru_way = 2*lru_way+lru_node[i];
    }

    return lru_way;
}

// called on every cache hit and cache fill
void CACHE::llc_update_replacement_state()
{
    // Backward track of the tree to flip the bits
    int node = 0;
    int mru_node[TREE_DEPTH];

    // Convert 'way' to binary and store the bits in the mru_node array
    for (int i=0; i<=TREE_DEPTH; i++) {
        mru_node[i] = (way >> i) & 1;
    }

    std::reverse(std::begin(mru_node), std::end(mru_node));

    // Flipping the bits of the traversed nodes
    for (int i=0; i<TREE_DEPTH; i++) {
        if (plru_tree[set][node] == mru_node[i]) {
            plru_tree[set][node] = !plru_tree[set][node]; // Flip the bit along the MRU path
        }
        node = 2*node+1+mru_node[i];
    }
}
```

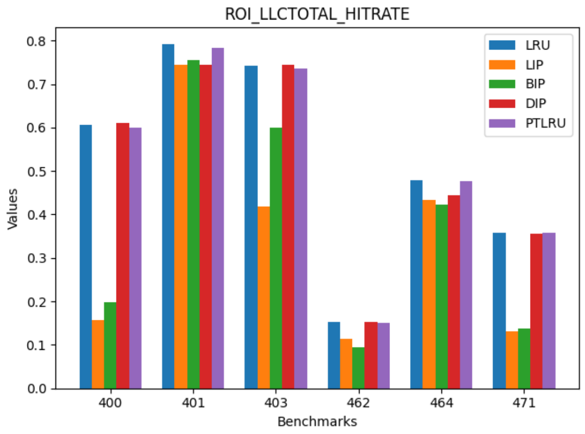
# 5 Geometric Mean IPC

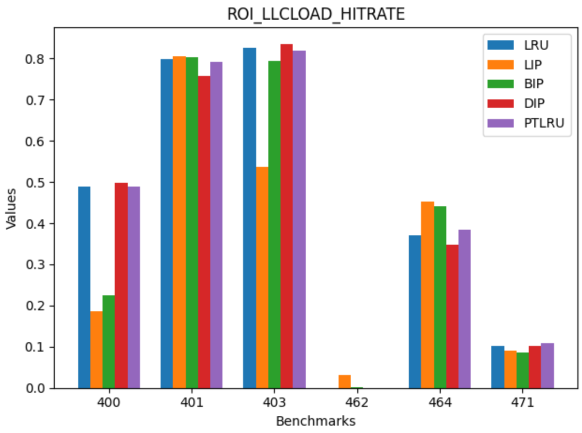
Compared against LRU.

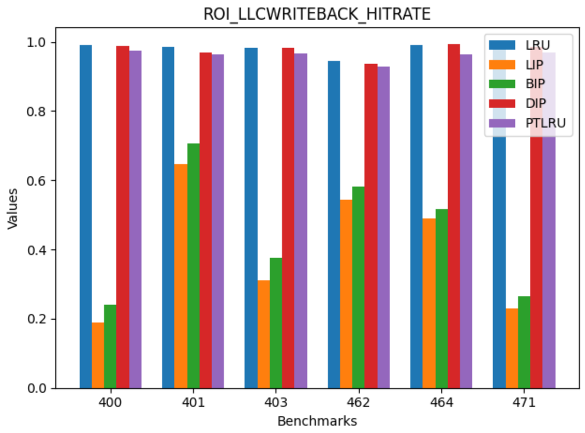
| LIP | BIP | DIP | PTLRU |
|---|---|---|---|
| 0.9637014210163327 | 0.9922082796596031 | 0.9931083108594528 | 1.001975287009498 |

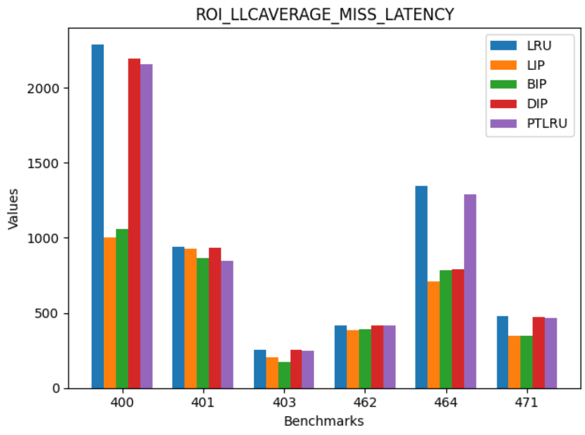
## 6 Benchmark Statistics


ROI_Cumulative_IPC
Values
Benchmarks
LRU
LIP
BIP
DIP
PTLRU


ROI_LLCTOTAL_HITRATE
Values
Benchmarks
LRU
LIP
BIP
DIP
PTLRU


ROI_LLCLOAD_HITRATE
Values
Benchmarks
LRU
LIP
BIP
DIP
PTLRU


ROI_LLCWRITEBACK_HITRATE
Values
Benchmarks
LRU
LIP
BIP
DIP
PTLRU


ROI_LLCAVERAGE_MISS_LATENCY
Values
Benchmarks
LRU
LIP
BIP
DIP
PTLRU