پروژهٔ نهایی درس معماری کامپیوتر
استاد: دکتر چشمی خانی

پیاده سازی Replacement Policy های مختلف و بررسی و مقایسهٔ کارایی آن ها به وسیلهٔ شبیه ساز ChampSim

نویسنده
مهدیار صلواتی
کد دانشجویی: ۴۰۲۲۴۳۰۸۰
ma.salavati@mail.sbu.ac.ir

چکیده

در این پروژه قصد داریم سیاست های جانشینی LRU، MRU LFU و FIFO را با استفاده از زبان C پیاده کرده و هر یک را با ۶۰ میلیون warmup instruction و ۴۰ میلیون simulate instruction شبیه سازی کرده و در نهایت پارامتر های Cumulative IPC - Hit Rate - Miss Rate را تفسیر و مقایسه کنیم. در تمامی تست ها از trace زیر استفاده شده است: bzip۲_۱۸۳B.trace.xz

۱ LRU

۱.۱ پیاده سازی الگوریتم

این سیاست جایگزینی بر اساس سن هر کدام از بلاک های کش کار می کند. یعنی وقتی کش پر شود و درخواست بلاک جدیدی دریافت شود، این سیاست بلاکی را اخراج می کند که از همه پیر تر یا به اصطلاح Least Recently Used است. منطقی است که در زمان hit شدن هر بلاک، آن بلاک به عنوان MRU شناخته شده و باقی بلاک ها پیر تر می شوند.

در اینجا ما کد C را بر اساس شماره clock که در آن قرار داریم پیاده می کنیم. یعنی یک آرایه به نام last_used_cycles به طول تعداد بلاک ها allocate کرده و به عنوان مقادیر اولیه صفر در درایه های آن وارد می کنیم.

همچنین سایز این آرایه که همان تعداد بلاک ها است را می توان این گونه محاسبه کرد که کش دارای x set است و هر ست دارای y way است. در نتیجه تعداد کل بلاک ها از حاصل ضرب x و y محاسبه می شود و در نهایت Constructor ما به این شکل در می آید:

```
myLRU::myLRU(...) {
    cycle_array_size = (size_t)(sets * ways);
    last_used_cycles = new uint64_t[cycle_array_size];
    for (size_t i = 0; i < cycle_array_size; i++) {
        last_used_cycles[i] = 0;
```

```
    }
}
```

با هر بار پر شدن یکی از way ها تابع replacement cache fill فراخوانی می شود. در این جا با توجه به توضیحاتی که دادیم، ابتدا شماره cycle که در آن قرار داریم را به عنوان درایهٔ آن way مورد نظر قرار داده و در نهایت cycle را increment می کنیم:

```
 void myLRU::replacement_cache_fill(...)
{
    last_used_cycles[(size_t)(set * NUM_WAY + way)] = cycle++;
}
```

شایان ذکر است که تابع بالا وقتی فراخوانی می شود که بلاک مورد نظر خالی بوده و یا محتوای جایگزین شده در آن miss شده بود و تابع update replacement state زمانی رخ می دهد که شاهد hit باشیم.

حال که معانی اعداد داخل آرایه را بررسی کردیم به راحتی می توانیم تابع find_victim را بررسی کنیم. این تابع بلاکی را اخراج می کند که last_used_cycles کمتری داشته باشد چرا که قدیمی تر است:

```
long myLRU::find_victim(...)
{
    long set_offset = set * NUM_WAY;
    long victim_way = 0;
    uint64_t min_cycle = last_used_cycles[set_offset];

    for (long i = 1; i < NUM_WAY; i++) {
        if (last_used_cycles[set_offset + i] < min_cycle) {
            min_cycle = last_used_cycles[set_offset + i];
            victim_way = i;
        }
    }

    assert(victim_way >= 0);
    assert(victim_way < NUM_WAY);
    return victim_way;
}
```

۲.۱ شبیه سازی

در این شبیه سازی و شبیه سازی های آتی پس از آپدیت کردن محتوای فایل ازchampsim_config.json دستور های زیر استفاده می کنیم:

```
./config.sh champsim_config.json
make
```

```
bin/champsim --warmup-instructions 60000000 --simulation-instructions
    40000000 /Users/mahdiyarsalavati/Downloads/bzip2_183B.trace.xz
```

نتایج خام شبیه سازی:

```
mahdiyarsalavati@Mahdiyars-MacBook-Air ChampSimLocal % bin/champsim --warmup-instructions 60000000 --
    simulation-instructions 40000000 /Users/mahdiyarsalavati/Downloads/bzip2_183B.trace.xz
[VMEM] WARNING: physical memory size is smaller than virtual memory size.

*** ChampSim Multicore Out-of-Order Simulator ***
Warmup Instructions: 60000000
Simulation Instructions: 40000000
Number of CPUs: 1
Page size: 4096

Off-chip DRAM Size: 16 GiB Channels: 1 Width: 64-bit Data Rate: 3205 MT/s
Heartbeat CPU 0 instructions: 10000003 cycles: 2573517 heartbeat IPC: 3.886 cumulative IPC: 3.886 (Simulation
    time: 00 hr 00 min 18 sec)
Heartbeat CPU 0 instructions: 20000005 cycles: 5086294 heartbeat IPC: 3.98 cumulative IPC: 3.932 (Simulation
    time: 00 hr 00 min 36 sec)
Heartbeat CPU 0 instructions: 30000007 cycles: 7597883 heartbeat IPC: 3.982 cumulative IPC: 3.948 (Simulation
    time: 00 hr 00 min 54 sec)
Heartbeat CPU 0 instructions: 40000008 cycles: 10110284 heartbeat IPC: 3.98 cumulative IPC: 3.956 (Simulation
    time: 00 hr 01 min 12 sec)
Heartbeat CPU 0 instructions: 50000008 cycles: 12624634 heartbeat IPC: 3.977 cumulative IPC: 3.961 (Simulation
     time: 00 hr 01 min 30 sec)
Warmup finished CPU 0 instructions: 60000000 cycles: 15140864 cumulative IPC: 3.963 (Simulation time: 00 hr 01
     min 48 sec)
Warmup complete CPU 0 instructions: 60000000 cycles: 15140864 cumulative IPC: 3.963 (Simulation time: 00 hr 01
     min 48 sec)
Heartbeat CPU 0 instructions: 60000008 cycles: 15140866 heartbeat IPC: 3.974 cumulative IPC: 4 (Simulation
    time: 00 hr 01 min 48 sec)
Heartbeat CPU 0 instructions: 70000010 cycles: 22365020 heartbeat IPC: 1.384 cumulative IPC: 1.384 (Simulation
     time: 00 hr 02 min 15 sec)
Heartbeat CPU 0 instructions: 80000010 cycles: 29110017 heartbeat IPC: 1.483 cumulative IPC: 1.432 (Simulation
     time: 00 hr 02 min 42 sec)
Heartbeat CPU 0 instructions: 90000010 cycles: 36163222 heartbeat IPC: 1.418 cumulative IPC: 1.427 (Simulation
     time: 00 hr 03 min 11 sec)
Simulation finished CPU 0 instructions: 40000001 cycles: 28788163 cumulative IPC: 1.389 (Simulation time: 00
    hr 03 min 40 sec)
Simulation complete CPU 0 instructions: 40000001 cycles: 28788163 cumulative IPC: 1.389 (Simulation time: 00
    hr 03 min 40 sec)

ChampSim completed all CPUs

=== Simulation ===
CPU 0 runs /Users/mahdiyarsalavati/Downloads/bzip2_183B.trace.xz

Region of Interest Statistics

CPU 0 cumulative IPC: 1.389 instructions: 40000001 cycles: 28788163
CPU 0 Branch Prediction Accuracy: 88.8% MPKI: 19.68 Average ROB Occupancy at Mispredict: 21.49
Branch type MPKI
BRANCH_DIRECT_JUMP: 0.0478
BRANCH_INDIRECT: 0
BRANCH_CONDITIONAL: 19.63
BRANCH_DIRECT_CALL: 0
```

```
BRANCH_INDIRECT_CALL: 0
BRANCH_RETURN: 0

cpu0->cpu0_STLB TOTAL        ACCESS:     214404 HIT:     214253 MISS:        151 MSHR_MERGE:          0
cpu0->cpu0_STLB LOAD         ACCESS:     214404 HIT:     214253 MISS:        151 MSHR_MERGE:          0
cpu0->cpu0_STLB RFO          ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_STLB PREFETCH     ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_STLB WRITE        ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_STLB TRANSLATION  ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_STLB PREFETCH REQUESTED:          0 ISSUED:          0 USEFUL:          0 USELESS:          0
cpu0->cpu0_STLB AVERAGE MISS LATENCY: 792 cycles
cpu0->cpu0_L2C TOTAL        ACCESS:     816970 HIT:     536435 MISS:     280535 MSHR_MERGE:          0
cpu0->cpu0_L2C LOAD         ACCESS:     480429 HIT:     315435 MISS:     164994 MSHR_MERGE:          0
cpu0->cpu0_L2C RFO          ACCESS:     157239 HIT:      41980 MISS:     115259 MSHR_MERGE:          0
cpu0->cpu0_L2C PREFETCH     ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_L2C WRITE        ACCESS:     179018 HIT:     178988 MISS:         30 MSHR_MERGE:          0
cpu0->cpu0_L2C TRANSLATION  ACCESS:        284 HIT:         32 MISS:        252 MSHR_MERGE:          0
cpu0->cpu0_L2C PREFETCH REQUESTED:          0 ISSUED:          0 USEFUL:          0 USELESS:          0
cpu0->cpu0_L2C AVERAGE MISS LATENCY: 89.21 cycles
cpu0->cpu0_L1I TOTAL        ACCESS:    1200398 HIT:    1200398 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_L1I LOAD         ACCESS:    1200398 HIT:    1200398 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_L1I RFO          ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_L1I PREFETCH     ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_L1I WRITE        ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_L1I TRANSLATION  ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_L1I PREFETCH REQUESTED:          0 ISSUED:          0 USEFUL:          0 USELESS:          0
cpu0->cpu0_L1I AVERAGE MISS LATENCY: - cycles
cpu0->cpu0_L1D TOTAL        ACCESS:   11538774 HIT:   10699096 MISS:     839678 MSHR_MERGE:     201724
cpu0->cpu0_L1D LOAD         ACCESS:    8591790 HIT:    7963728 MISS:     628062 MSHR_MERGE:     147632
cpu0->cpu0_L1D RFO          ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_L1D PREFETCH     ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_L1D WRITE        ACCESS:    2946682 HIT:    2735358 MISS:     211324 MSHR_MERGE:      54084
cpu0->cpu0_L1D TRANSLATION  ACCESS:        302 HIT:         10 MISS:        292 MSHR_MERGE:          8
cpu0->cpu0_L1D PREFETCH REQUESTED:          0 ISSUED:          0 USEFUL:          0 USELESS:          0
cpu0->cpu0_L1D AVERAGE MISS LATENCY: 47.97 cycles
cpu0->cpu0_ITLB TOTAL        ACCESS:     960937 HIT:     960937 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_ITLB LOAD         ACCESS:     960937 HIT:     960937 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_ITLB RFO          ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_ITLB PREFETCH     ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_ITLB WRITE        ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_ITLB TRANSLATION  ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_ITLB PREFETCH REQUESTED:          0 ISSUED:          0 USEFUL:          0 USELESS:          0
cpu0->cpu0_ITLB AVERAGE MISS LATENCY: - cycles
cpu0->cpu0_DTLB TOTAL        ACCESS:   10984452 HIT:   10729968 MISS:     254484 MSHR_MERGE:      40081
cpu0->cpu0_DTLB LOAD         ACCESS:   10984452 HIT:   10729968 MISS:     254484 MSHR_MERGE:      40081
cpu0->cpu0_DTLB RFO          ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_DTLB PREFETCH     ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_DTLB WRITE        ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_DTLB TRANSLATION  ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_DTLB PREFETCH REQUESTED:          0 ISSUED:          0 USEFUL:          0 USELESS:          0
cpu0->cpu0_DTLB AVERAGE MISS LATENCY: 5.559 cycles
cpu0->LLC TOTAL        ACCESS:     409061 HIT:     345389 MISS:      63672 MSHR_MERGE:          0
cpu0->LLC LOAD         ACCESS:     164994 HIT:     135503 MISS:      29491 MSHR_MERGE:          0
cpu0->LLC RFO          ACCESS:     115259 HIT:      81332 MISS:      33927 MSHR_MERGE:          0
cpu0->LLC PREFETCH     ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->LLC WRITE        ACCESS:     128556 HIT:     128553 MISS:          3 MSHR_MERGE:          0
cpu0->LLC TRANSLATION  ACCESS:        252 HIT:          1 MISS:        251 MSHR_MERGE:          0
cpu0->LLC PREFETCH REQUESTED:          0 ISSUED:          0 USEFUL:          0 USELESS:          0
```

```
cpu0->LLC AVERAGE MISS LATENCY: 325.7 cycles

DRAM Statistics

Channel 0 RQ ROW_BUFFER_HIT:       2779
  ROW_BUFFER_MISS:      60613
  AVG DBUS CONGESTED CYCLE: 12.16
Channel 0 WQ ROW_BUFFER_HIT:       9245
  ROW_BUFFER_MISS:      37224
  FULL:          0
Channel 0 REFRESHES ISSUED:       2399
```

حال با توجه به دو فرمول زیر پارامتر های خواسته شده را بدست می آوریم:

$$\text{Miss Rate} = \frac{\#\text{ of misses}}{\#\text{ of total accesses}} \times 100 = \frac{63672}{409061} \times 100 = \boxed{15.57\%}$$

$$\text{Hit Rate} = \frac{\#\text{ of hits}}{\#\text{ of total accesses}} \times 100 = \frac{345389}{409061} \times 100 = \boxed{84.43\%}$$

$$\text{Cumulative IPC} = \boxed{1.389}$$

توضیحات مربوط به نرخ تصادم و نرخ miss به صورت فرمول ریاضی داده شد. پارامتر Cumulative IPC نشان دهندهٔ تعداد Instruction هایی است که در یک کلاک می توانند انجام شوند. کلمهٔ Cumulative به این معناست که عدد نهایی تجمیعی از IPC هایی است که در طول شبیه سازی با آنها مواجه بودیم. برای مثال شبیه سازی با Heartbeat IPC برابر ۳.۹۷۴ شروع به کار کرد اما با پیشرفت در شبیه سازی این مقدار کمتر شد.

به طور کلی می توان گفت این سیاست عملکرد مناسبی داشته و شاهد Rate Hit بالایی هستیم و مقدار Instrucion Per Clock نیز عدد مطلوبی به نظر می رسد. در نهایت با مقایسهٔ این چهار روش، به دید بهتری از این اعداد خواهیم رسید.

LFU ۲

۱.۲ پیاده سازی الگوریتم

این سیاست جایگزینی بر اساس فراوانی دسترسی ها به بلاک های مختلف عمل می کند. بر این اساس که دو آرایه نگه می داریم که هر دو به تعداد بلاک های کش هستند. آرایهٔ اولی مشابه LRU برای هر بلاک، آخرین کلاکی را نگه می دارد که به آن بلاک دسترسی داشته ایم. اسم این آرایه مانند گذشته last_used_cycles است. اما آرایهٔ دوم همان چیزی است که این سیاست را از LRU متمایز می کند. این آرایه که از اسمش frequency_counters پیداست قرار است تعداد دفعاتی را نگه دارد که هر یک از بلاک های کش فراخوانی شده اند.

در نهایت در زمان پر شدن کش، برای اخراج قربانی به این شکل عمل می کند که ابتدا بلاکی(هایی) را پیدا می کند که frequency_counters آن ها مینیمم است. در نتیجه کمتر فراخوانی شده اند و آنها نامزد اول اخراج هستند. سپس پس از یافت آنها، آن بلاکی را اخراج می کند که از همه قدیمی تر است (یعنی آن که last_used_cycles کمتری دارد).

پس با توجه به توضیحات داده شده، آرایه ها در صورتی که کش خالی باشد این گونه پر می شوند:

```
void myLFU::replacement_cache_fill(...)
{
    size_t block_index = (size_t)(set * NUM_WAY + way);
    frequency_counters[block_index] = 1;
    last_used_cycles[block_index] = cycle++;
}
```

و در صورتی که شاهد hit باشیم این گونه محتوای آرایه ها تغییر می کند:

```
void myLFU::update_replacement_state(...)
{
    if (hit && access_type{type} != access_type::WRITE) {
        size_t block_index = (size_t)(set * NUM_WAY + way);
        frequency_counters[block_index]++;
        last_used_cycles[block_index] = cycle++;
    }
}
```

حال بر اساس منطق توضیح داده شده، در تابع find_victim بلاکی که کمترین frequency و سپس کمترین سن را دارد اخراج می شود:

```
long myLFU::find_victim(...)
{
    long set_offset = set * NUM_WAY;

    uint64_t min_freq = frequency_counters[set_offset];
    for (long i = 1; i < NUM_WAY; i++) {
        if (frequency_counters[set_offset + i] < min_freq) {
            min_freq = frequency_counters[set_offset + i];
        }
    }

    long victim_way = -1;
    uint64_t min_cycle = std::numeric_limits<uint64_t>::max();
    for (long i = 0; i < NUM_WAY; i++) {
        if (frequency_counters[set_offset + i] == min_freq) {
            if (last_used_cycles[set_offset + i] < min_cycle) {
                min_cycle = last_used_cycles[set_offset + i];
                victim_way = i;
            }
        }
```

```
    }

    assert(victim_way != -1);
    return victim_way;
}
```

۲.۲ شبیه سازی

نتایج خام شبیه سازی:

```
   mahdiyarsalavati@Mahdiyars-MacBook-Air ChampSimLocal % bin/champsim --warmup-instructions 60000000 --
        simulation-instructions 40000000 /Users/mahdiyarsalavati/Downloads/bzip2_183B.trace.xz
[VMEM] WARNING: physical memory size is smaller than virtual memory size.

*** ChampSim Multicore Out-of-Order Simulator ***
Warmup Instructions: 60000000
Simulation Instructions: 40000000
Number of CPUs: 1
Page size: 4096

Off-chip DRAM Size: 16 GiB Channels: 1 Width: 64-bit Data Rate: 3205 MT/s
Heartbeat CPU 0 instructions: 10000003 cycles: 2573507 heartbeat IPC: 3.886 cumulative IPC: 3.886 (Simulation
     time: 00 hr 00 min 18 sec)
Heartbeat CPU 0 instructions: 20000005 cycles: 5086286 heartbeat IPC: 3.98 cumulative IPC: 3.932 (Simulation
     time: 00 hr 00 min 36 sec)
Heartbeat CPU 0 instructions: 30000007 cycles: 7597843 heartbeat IPC: 3.982 cumulative IPC: 3.948 (Simulation
     time: 00 hr 00 min 54 sec)
Heartbeat CPU 0 instructions: 40000008 cycles: 10110248 heartbeat IPC: 3.98 cumulative IPC: 3.956 (Simulation
     time: 00 hr 01 min 12 sec)
Heartbeat CPU 0 instructions: 50000008 cycles: 12624617 heartbeat IPC: 3.977 cumulative IPC: 3.961 (Simulation
      time: 00 hr 01 min 30 sec)
Warmup finished CPU 0 instructions: 60000000 cycles: 15140856 cumulative IPC: 3.963 (Simulation time: 00 hr 01
      min 47 sec)
Warmup complete CPU 0 instructions: 60000000 cycles: 15140856 cumulative IPC: 3.963 (Simulation time: 00 hr 01
      min 47 sec)
Heartbeat CPU 0 instructions: 60000008 cycles: 15140858 heartbeat IPC: 3.974 cumulative IPC: 4 (Simulation
     time: 00 hr 01 min 47 sec)
Heartbeat CPU 0 instructions: 70000010 cycles: 22539981 heartbeat IPC: 1.352 cumulative IPC: 1.352 (Simulation
      time: 00 hr 02 min 16 sec)
Heartbeat CPU 0 instructions: 80000010 cycles: 29384749 heartbeat IPC: 1.461 cumulative IPC: 1.404 (Simulation
      time: 00 hr 02 min 43 sec)
Heartbeat CPU 0 instructions: 90000010 cycles: 36367405 heartbeat IPC: 1.432 cumulative IPC: 1.413 (Simulation
      time: 00 hr 03 min 11 sec)
Simulation finished CPU 0 instructions: 40000001 cycles: 29309099 cumulative IPC: 1.365 (Simulation time: 00
     hr 03 min 41 sec)
Simulation complete CPU 0 instructions: 40000001 cycles: 29309099 cumulative IPC: 1.365 (Simulation time: 00
     hr 03 min 41 sec)

ChampSim completed all CPUs

=== Simulation ===
CPU 0 runs /Users/mahdiyarsalavati/Downloads/bzip2_183B.trace.xz

Region of Interest Statistics

```

```
CPU 0 cumulative IPC: 1.365 instructions: 40000001 cycles: 29309099
CPU 0 Branch Prediction Accuracy: 88.8% MPKI: 19.68 Average ROB Occupancy at Mispredict: 21.38
Branch type MPKI
BRANCH_DIRECT_JUMP: 0.0478
BRANCH_INDIRECT: 0
BRANCH_CONDITIONAL: 19.63
BRANCH_DIRECT_CALL: 0
BRANCH_INDIRECT_CALL: 0
BRANCH_RETURN: 0

cpu0->cpu0_STLB TOTAL        ACCESS:     214402 HIT:     214251 MISS:        151 MSHR_MERGE:          0
cpu0->cpu0_STLB LOAD         ACCESS:     214402 HIT:     214251 MISS:        151 MSHR_MERGE:          0
cpu0->cpu0_STLB RFO          ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_STLB PREFETCH     ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_STLB WRITE        ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_STLB TRANSLATION  ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_STLB PREFETCH REQUESTED:          0 ISSUED:          0 USEFUL:          0 USELESS:          0
cpu0->cpu0_STLB AVERAGE MISS LATENCY: 776.1 cycles
cpu0->cpu0_L2C TOTAL        ACCESS:     817041 HIT:     536498 MISS:     280543 MSHR_MERGE:          0
cpu0->cpu0_L2C LOAD         ACCESS:     480431 HIT:     315460 MISS:     164971 MSHR_MERGE:          0
cpu0->cpu0_L2C RFO          ACCESS:     157274 HIT:      41984 MISS:     115290 MSHR_MERGE:          0
cpu0->cpu0_L2C PREFETCH     ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_L2C WRITE        ACCESS:     179052 HIT:     179022 MISS:         30 MSHR_MERGE:          0
cpu0->cpu0_L2C TRANSLATION  ACCESS:        284 HIT:         32 MISS:        252 MSHR_MERGE:          0
cpu0->cpu0_L2C PREFETCH REQUESTED:          0 ISSUED:          0 USEFUL:          0 USELESS:          0
cpu0->cpu0_L2C AVERAGE MISS LATENCY: 71.33 cycles
cpu0->cpu0_L1I TOTAL        ACCESS:    1206692 HIT:    1206692 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_L1I LOAD         ACCESS:    1206692 HIT:    1206692 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_L1I RFO          ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_L1I PREFETCH     ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_L1I WRITE        ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_L1I TRANSLATION  ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_L1I PREFETCH REQUESTED:          0 ISSUED:          0 USEFUL:          0 USELESS:          0
cpu0->cpu0_L1I AVERAGE MISS LATENCY: - cycles
cpu0->cpu0_L1D TOTAL        ACCESS:   11561545 HIT:   10716752 MISS:     844793 MSHR_MERGE:     206803
cpu0->cpu0_L1D LOAD         ACCESS:    8614517 HIT:    7971074 MISS:     643443 MSHR_MERGE:     163011
cpu0->cpu0_L1D RFO          ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_L1D PREFETCH     ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_L1D WRITE        ACCESS:    2946726 HIT:    2745668 MISS:     201058 MSHR_MERGE:      43784
cpu0->cpu0_L1D TRANSLATION  ACCESS:        302 HIT:         10 MISS:        292 MSHR_MERGE:          8
cpu0->cpu0_L1D PREFETCH REQUESTED:          0 ISSUED:          0 USEFUL:          0 USELESS:          0
cpu0->cpu0_L1D AVERAGE MISS LATENCY: 40.07 cycles
cpu0->cpu0_ITLB TOTAL        ACCESS:     965820 HIT:     965820 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_ITLB LOAD         ACCESS:     965820 HIT:     965820 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_ITLB RFO          ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_ITLB PREFETCH     ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_ITLB WRITE        ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_ITLB TRANSLATION  ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_ITLB PREFETCH REQUESTED:          0 ISSUED:          0 USEFUL:          0 USELESS:          0
cpu0->cpu0_ITLB AVERAGE MISS LATENCY: - cycles
cpu0->cpu0_DTLB TOTAL        ACCESS:   11006398 HIT:   10752022 MISS:     254376 MSHR_MERGE:      39975
cpu0->cpu0_DTLB LOAD         ACCESS:   11006398 HIT:   10752022 MISS:     254376 MSHR_MERGE:      39975
cpu0->cpu0_DTLB RFO          ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_DTLB PREFETCH     ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_DTLB WRITE        ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_DTLB TRANSLATION  ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_DTLB PREFETCH REQUESTED:          0 ISSUED:          0 USEFUL:          0 USELESS:          0
cpu0->cpu0_DTLB AVERAGE MISS LATENCY: 5.548 cycles
```

```
cpu0->LLC TOTAL        ACCESS:     409101 HIT:     314248 MISS:      94853 MSHR_MERGE:          0
cpu0->LLC LOAD         ACCESS:     164971 HIT:     134937 MISS:      30034 MSHR_MERGE:          0
cpu0->LLC RFO          ACCESS:     115289 HIT:      83861 MISS:      31428 MSHR_MERGE:          0
cpu0->LLC PREFETCH     ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->LLC WRITE        ACCESS:     128589 HIT:      95449 MISS:      33140 MSHR_MERGE:          0
cpu0->LLC TRANSLATION  ACCESS:        252 HIT:          1 MISS:        251 MSHR_MERGE:          0
cpu0->LLC PREFETCH REQUESTED:          0 ISSUED:          0 USEFUL:          0 USELESS:          0
cpu0->LLC AVERAGE MISS LATENCY: 169 cycles

DRAM Statistics

Channel 0 RQ ROW_BUFFER_HIT:       2039
  ROW_BUFFER_MISS:      59346
  AVG DBUS CONGESTED CYCLE: 13.18
Channel 0 WQ ROW_BUFFER_HIT:       9825
  ROW_BUFFER_MISS:      31625
  FULL:        130
Channel 0 REFRESHES ISSUED:       2443
```

$$\text{Miss Rate} = \frac{\#\text{ of misses}}{\#\text{ of total accesses}} \times 100 = \frac{94853}{409101} \times 100 = \boxed{23.2\%}$$

$$\text{Hit Rate} = \frac{\#\text{ of hits}}{\#\text{ of total accesses}} \times 100 = \frac{314248}{409101} \times 100 = \boxed{76.8\%}$$

$$\text{Cumulative IPC} = \boxed{1.365}$$

همان طور که می بینیم با آنکه شباهت زیادی هم در دو الگوریتم LRU ، LFU و هم در نتایج آن وجود دارد اما می توان دید عملکرد LFU کمی بد تر از LRU بوده است. می توان ریشهٔ این رخداد را در پیچیده تر بودن سیاست LFU دانست چرا که در تابع find_victim تقریبا به اندازهٔ دو برابر تعداد مقایسه هایی که در LRU انجام می شد، مقایسه باید انجام دهیم. البته که در برخی از ترکیب های دیگر Instruction ها سیاست LFU می تواند منجر به up speed قابل توجهی شود اما در این ترکیب از Instruction ها شاهد چنین بهبودی نبودیم.

MRU ۳

۱.۳ پیاده سازی الگوریتم

این سیاست در پیاده سازی بسیار مشابه LRU است با این تفاوت که در تابع find_victim آن باید ماکزیمم آرایهٔ last_used_cycles را به عنوان قربانی معرفی کنیم:

```
long myMRU::find_victim(...)
{
    long set_offset = set * NUM_WAY;
    long victim_way = 0;
```

```
    uint64_t max_cycle = last_used_cycles[set_offset];

    for (long i = 1; i < NUM_WAY; i++) {
        if (last_used_cycles[set_offset + i] > max_cycle) {
            max_cycle = last_used_cycles[set_offset + i];
            victim_way = i;
        }
    }

    assert(victim_way >= 0);
    assert(victim_way < NUM_WAY);
    return victim_way;
}
```

۲.۳ شبیه سازی

نتایج خام شبیه سازی:

```
mahdiyarsalavati@Mahdiyars-MacBook-Air ChampSimLocal % bin/champsim --warmup-instructions 60000000 --
    simulation-instructions 40000000 /Users/mahdiyarsalavati/Downloads/bzip2_183B.trace.xz
[VMEM] WARNING: physical memory size is smaller than virtual memory size.

*** ChampSim Multicore Out-of-Order Simulator ***
Warmup Instructions: 60000000
Simulation Instructions: 40000000
Number of CPUs: 1
Page size: 4096

Off-chip DRAM Size: 16 GiB Channels: 1 Width: 64-bit Data Rate: 3205 MT/s
Heartbeat CPU 0 instructions: 10000000 cycles: 2573519 heartbeat IPC: 3.886 cumulative IPC: 3.886 (Simulation
    time: 00 hr 00 min 18 sec)
Heartbeat CPU 0 instructions: 20000001 cycles: 5086305 heartbeat IPC: 3.98 cumulative IPC: 3.932 (Simulation
    time: 00 hr 00 min 36 sec)
Heartbeat CPU 0 instructions: 30000003 cycles: 7597899 heartbeat IPC: 3.982 cumulative IPC: 3.948 (Simulation
    time: 00 hr 00 min 54 sec)
Heartbeat CPU 0 instructions: 40000003 cycles: 10110316 heartbeat IPC: 3.98 cumulative IPC: 3.956 (Simulation
    time: 00 hr 01 min 12 sec)
Heartbeat CPU 0 instructions: 50000004 cycles: 12624668 heartbeat IPC: 3.977 cumulative IPC: 3.961 (Simulation
     time: 00 hr 01 min 30 sec)
Warmup finished CPU 0 instructions: 60000000 cycles: 15140901 cumulative IPC: 3.963 (Simulation time: 00 hr 01
     min 48 sec)
Warmup complete CPU 0 instructions: 60000000 cycles: 15140901 cumulative IPC: 3.963 (Simulation time: 00 hr 01
     min 48 sec)
Heartbeat CPU 0 instructions: 60000004 cycles: 15140902 heartbeat IPC: 3.974 cumulative IPC: 4 (Simulation
    time: 00 hr 01 min 48 sec)
Heartbeat CPU 0 instructions: 70000005 cycles: 24786180 heartbeat IPC: 1.037 cumulative IPC: 1.037 (Simulation
     time: 00 hr 02 min 25 sec)
Heartbeat CPU 0 instructions: 80000005 cycles: 33857831 heartbeat IPC: 1.102 cumulative IPC: 1.069 (Simulation
     time: 00 hr 03 min 00 sec)
Heartbeat CPU 0 instructions: 90000007 cycles: 43315911 heartbeat IPC: 1.057 cumulative IPC: 1.065 (Simulation
     time: 00 hr 03 min 38 sec)
```

```
Simulation finished CPU 0 instructions: 40000001 cycles: 39046067 cumulative IPC: 1.024 (Simulation time: 00 hr 04 min 19 sec)
Simulation complete CPU 0 instructions: 40000001 cycles: 39046067 cumulative IPC: 1.024 (Simulation time: 00 hr 04 min 19 sec)

ChampSim completed all CPUs

=== Simulation ===
CPU 0 runs /Users/mahdiyarsalavati/Downloads/bzip2_183B.trace.xz

Region of Interest Statistics

CPU 0 cumulative IPC: 1.024 instructions: 40000001 cycles: 39046067
CPU 0 Branch Prediction Accuracy: 88.8% MPKI: 19.68 Average ROB Occupancy at Mispredict: 21.95
Branch type MPKI
BRANCH_DIRECT_JUMP: 0.0478
BRANCH_INDIRECT: 0
BRANCH_CONDITIONAL: 19.63
BRANCH_DIRECT_CALL: 0
BRANCH_INDIRECT_CALL: 0
BRANCH_RETURN: 0

cpu0->cpu0_STLB TOTAL        ACCESS:     214402 HIT:     214251 MISS:        151 MSHR_MERGE:          0
cpu0->cpu0_STLB LOAD         ACCESS:     214402 HIT:     214251 MISS:        151 MSHR_MERGE:          0
cpu0->cpu0_STLB RFO          ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_STLB PREFETCH     ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_STLB WRITE        ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_STLB TRANSLATION  ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_STLB PREFETCH REQUESTED:          0 ISSUED:          0 USEFUL:          0 USELESS:          0
cpu0->cpu0_STLB AVERAGE MISS LATENCY: 828.8 cycles
cpu0->cpu0_L2C TOTAL        ACCESS:     817042 HIT:     536528 MISS:     280514 MSHR_MERGE:          0
cpu0->cpu0_L2C LOAD         ACCESS:     480428 HIT:     315476 MISS:     164952 MSHR_MERGE:          0
cpu0->cpu0_L2C RFO          ACCESS:     157275 HIT:      41995 MISS:     115280 MSHR_MERGE:          0
cpu0->cpu0_L2C PREFETCH     ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_L2C WRITE        ACCESS:     179055 HIT:     179025 MISS:         30 MSHR_MERGE:          0
cpu0->cpu0_L2C TRANSLATION  ACCESS:        284 HIT:         32 MISS:        252 MSHR_MERGE:          0
cpu0->cpu0_L2C PREFETCH REQUESTED:          0 ISSUED:          0 USEFUL:          0 USELESS:          0
cpu0->cpu0_L2C AVERAGE MISS LATENCY: 195.2 cycles
cpu0->cpu0_L1I TOTAL        ACCESS:    1233013 HIT:    1233013 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_L1I LOAD         ACCESS:    1233013 HIT:    1233013 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_L1I RFO          ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_L1I PREFETCH     ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_L1I WRITE        ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_L1I TRANSLATION  ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_L1I PREFETCH REQUESTED:          0 ISSUED:          0 USEFUL:          0 USELESS:          0
cpu0->cpu0_L1I AVERAGE MISS LATENCY: - cycles
cpu0->cpu0_L1D TOTAL        ACCESS:   11544491 HIT:   10731017 MISS:     813474 MSHR_MERGE:     175484
cpu0->cpu0_L1D LOAD         ACCESS:    8597481 HIT:    7968727 MISS:     628754 MSHR_MERGE:     148325
cpu0->cpu0_L1D RFO          ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_L1D PREFETCH     ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_L1D WRITE        ACCESS:    2946708 HIT:    2762280 MISS:     184428 MSHR_MERGE:      27151
cpu0->cpu0_L1D TRANSLATION  ACCESS:        302 HIT:         10 MISS:        292 MSHR_MERGE:          8
cpu0->cpu0_L1D PREFETCH REQUESTED:          0 ISSUED:          0 USEFUL:          0 USELESS:          0
cpu0->cpu0_L1D AVERAGE MISS LATENCY: 94.52 cycles
cpu0->cpu0_ITLB TOTAL        ACCESS:     988104 HIT:     988104 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_ITLB LOAD         ACCESS:     988104 HIT:     988104 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_ITLB RFO          ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_ITLB PREFETCH     ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
```

```
cpu0->cpu0_ITLB WRITE        ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_ITLB TRANSLATION  ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_ITLB PREFETCH REQUESTED:          0 ISSUED:          0 USEFUL:          0 USELESS:          0
cpu0->cpu0_ITLB AVERAGE MISS LATENCY: - cycles
cpu0->cpu0_DTLB TOTAL        ACCESS:   10991125 HIT:   10737610 MISS:     253515 MSHR_MERGE:      39114
cpu0->cpu0_DTLB LOAD         ACCESS:   10991125 HIT:   10737610 MISS:     253515 MSHR_MERGE:      39114
cpu0->cpu0_DTLB RFO          ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_DTLB PREFETCH     ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_DTLB WRITE        ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_DTLB TRANSLATION  ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_DTLB PREFETCH REQUESTED:          0 ISSUED:          0 USEFUL:          0 USELESS:          0
cpu0->cpu0_DTLB AVERAGE MISS LATENCY: 5.585 cycles
cpu0->LLC TOTAL        ACCESS:     409072 HIT:     154771 MISS:     254301 MSHR_MERGE:          0
cpu0->LLC LOAD         ACCESS:     164952 HIT:      50233 MISS:     114719 MSHR_MERGE:          0
cpu0->LLC RFO          ACCESS:     115280 HIT:      58241 MISS:      57039 MSHR_MERGE:          0
cpu0->LLC PREFETCH     ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->LLC WRITE        ACCESS:     128588 HIT:      46296 MISS:      82292 MSHR_MERGE:          0
cpu0->LLC TRANSLATION  ACCESS:        252 HIT:          1 MISS:        251 MSHR_MERGE:          0
cpu0->LLC PREFETCH REQUESTED:          0 ISSUED:          0 USEFUL:          0 USELESS:          0
cpu0->LLC AVERAGE MISS LATENCY: 201 cycles

DRAM Statistics

Channel 0 RQ ROW_BUFFER_HIT:       4728
  ROW_BUFFER_MISS:     167088
  AVG DBUS CONGESTED CYCLE: 8.784
Channel 0 WQ ROW_BUFFER_HIT:      22917
  ROW_BUFFER_MISS:      81957
  FULL:        378
Channel 0 REFRESHES ISSUED:       3254
```

$$\text{Miss Rate} = \frac{\#\text{ of misses}}{\#\text{ of total accesses}} \times 100 = \frac{254301}{409072} \times 100 = \boxed{62.16\%}$$

$$\text{Hit Rate} = \frac{\#\text{ of hits}}{\#\text{ of total accesses}} \times 100 = \frac{154771}{409072} \times 100 = \boxed{37.84\%}$$

$$\text{Cumulative IPC} = \boxed{1.024}$$

همان طور که انتظارش را داشتیم و می دانستیم این سیاست spatial locality-unfriendly است، به Hit Rate و IPC به شدت کمتری منجر شد.

FIFO ۴

۱.۴ پیاده سازی الگوریتم

این سیاست بلاک اخراجی را بر اساس یک صف پیدا می کند و می توان گفت تفاوت اصلی آن با LRU در این است که زمان hit هیچ عملی رخ نمی دهد.

پس تابع find_victim آن تفاوت چندانی با LRU ندارد و باز هم باید در آرایه کمترین عضو را اخراج کنیم:

```
long myFIFO::find_victim(...)
{
    long set_offset = set * NUM_WAY;
    long victim_way = 0;
    uint64_t min_cycle = arrival_cycles[set_offset];

    for (long i = 1; i < NUM_WAY; i++) {
        if (arrival_cycles[set_offset + i] < min_cycle) {
            min_cycle = arrival_cycles[set_offset + i];
            victim_way = i;
        }
    }

    assert(victim_way >= 0);
    assert(victim_way < NUM_WAY);
    return victim_way;
}
```

پس تابع update_replacement_state عملا خالی می شود و سایر توابع که اشاره نشده اند هم نسبت به LRU بی تفاوت باقی می مانند:

```
void myFIFO::update_replacement_state(...)
{

}
```

۲.۴ شبیه سازی

نتایج خام شبیه سازی:

```
mahdiyarsalavati@Mahdiyars-MacBook-Air ChampSimLocal % bin/champsim --warmup-instructions 60000000 --
    simulation-instructions 40000000 /Users/mahdiyarsalavati/Downloads/bzip2_183B.trace.xz
[VMEM] WARNING: physical memory size is smaller than virtual memory size.

*** ChampSim Multicore Out-of-Order Simulator ***
Warmup Instructions: 60000000
Simulation Instructions: 40000000
Number of CPUs: 1
Page size: 4096

Off-chip DRAM Size: 16 GiB Channels: 1 Width: 64-bit Data Rate: 3205 MT/s
Heartbeat CPU 0 instructions: 10000000 cycles: 2573516 heartbeat IPC: 3.886 cumulative IPC: 3.886 (Simulation
    time: 00 hr 00 min 18 sec)
Heartbeat CPU 0 instructions: 20000001 cycles: 5086295 heartbeat IPC: 3.98 cumulative IPC: 3.932 (Simulation
    time: 00 hr 00 min 35 sec)
```

```
Heartbeat CPU 0 instructions: 30000003 cycles: 7597874 heartbeat IPC: 3.982 cumulative IPC: 3.948 (Simulation time: 00 hr 00 min 53 sec)
Heartbeat CPU 0 instructions: 40000003 cycles: 10110279 heartbeat IPC: 3.98 cumulative IPC: 3.956 (Simulation time: 00 hr 01 min 12 sec)
Heartbeat CPU 0 instructions: 50000004 cycles: 12624616 heartbeat IPC: 3.977 cumulative IPC: 3.961 (Simulation time: 00 hr 01 min 29 sec)
Warmup finished CPU 0 instructions: 60000000 cycles: 15140851 cumulative IPC: 3.963 (Simulation time: 00 hr 01 min 47 sec)
Warmup complete CPU 0 instructions: 60000000 cycles: 15140851 cumulative IPC: 3.963 (Simulation time: 00 hr 01 min 47 sec)
Heartbeat CPU 0 instructions: 60000004 cycles: 15140852 heartbeat IPC: 3.974 cumulative IPC: 4 (Simulation time: 00 hr 01 min 47 sec)
Heartbeat CPU 0 instructions: 70000005 cycles: 22857611 heartbeat IPC: 1.296 cumulative IPC: 1.296 (Simulation time: 00 hr 02 min 16 sec)
Heartbeat CPU 0 instructions: 80000005 cycles: 30052388 heartbeat IPC: 1.39 cumulative IPC: 1.341 (Simulation time: 00 hr 02 min 45 sec)
Heartbeat CPU 0 instructions: 90000007 cycles: 37556448 heartbeat IPC: 1.333 cumulative IPC: 1.338 (Simulation time: 00 hr 03 min 15 sec)
Simulation finished CPU 0 instructions: 40000001 cycles: 30711843 cumulative IPC: 1.302 (Simulation time: 00 hr 03 min 46 sec)
Simulation complete CPU 0 instructions: 40000001 cycles: 30711843 cumulative IPC: 1.302 (Simulation time: 00 hr 03 min 46 sec)

ChampSim completed all CPUs

=== Simulation ===
CPU 0 runs /Users/mahdiyarsalavati/Downloads/bzip2_183B.trace.xz

Region of Interest Statistics

CPU 0 cumulative IPC: 1.302 instructions: 40000001 cycles: 30711843
CPU 0 Branch Prediction Accuracy: 88.8% MPKI: 19.68 Average ROB Occupancy at Mispredict: 21.57
Branch type MPKI
BRANCH_DIRECT_JUMP: 0.0478
BRANCH_INDIRECT: 0
BRANCH_CONDITIONAL: 19.63
BRANCH_DIRECT_CALL: 0
BRANCH_INDIRECT_CALL: 0
BRANCH_RETURN: 0

cpu0->cpu0_STLB TOTAL        ACCESS:     214421 HIT:     214270 MISS:        151 MSHR_MERGE:          0
cpu0->cpu0_STLB LOAD         ACCESS:     214421 HIT:     214270 MISS:        151 MSHR_MERGE:          0
cpu0->cpu0_STLB RFO          ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_STLB PREFETCH     ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_STLB WRITE        ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_STLB TRANSLATION  ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_STLB PREFETCH REQUESTED:          0 ISSUED:          0 USEFUL:          0 USELESS:          0
cpu0->cpu0_STLB AVERAGE MISS LATENCY: 790.3 cycles
cpu0->cpu0_L2C TOTAL        ACCESS:     816986 HIT:     536439 MISS:     280547 MSHR_MERGE:          0
cpu0->cpu0_L2C LOAD         ACCESS:     480430 HIT:     315429 MISS:     165001 MSHR_MERGE:          0
cpu0->cpu0_L2C RFO          ACCESS:     157246 HIT:      41982 MISS:     115264 MSHR_MERGE:          0
cpu0->cpu0_L2C PREFETCH     ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_L2C WRITE        ACCESS:     179026 HIT:     178996 MISS:         30 MSHR_MERGE:          0
cpu0->cpu0_L2C TRANSLATION  ACCESS:        284 HIT:         32 MISS:        252 MSHR_MERGE:          0
cpu0->cpu0_L2C PREFETCH REQUESTED:          0 ISSUED:          0 USEFUL:          0 USELESS:          0
cpu0->cpu0_L2C AVERAGE MISS LATENCY: 108.3 cycles
cpu0->cpu0_L1I TOTAL        ACCESS:    1207573 HIT:    1207573 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_L1I LOAD         ACCESS:    1207573 HIT:    1207573 MISS:          0 MSHR_MERGE:          0
```

```
cpu0->cpu0_L1I RFO          ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_L1I PREFETCH     ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_L1I WRITE        ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_L1I TRANSLATION  ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_L1I PREFETCH REQUESTED:          0 ISSUED:          0 USEFUL:          0 USELESS:          0
cpu0->cpu0_L1I AVERAGE MISS LATENCY: - cycles
cpu0->cpu0_L1D TOTAL        ACCESS:   11529610 HIT:   10702986 MISS:     826624 MSHR_MERGE:     188662
cpu0->cpu0_L1D LOAD         ACCESS:    8582608 HIT:    7960590 MISS:     622018 MSHR_MERGE:     141587
cpu0->cpu0_L1D RFO          ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_L1D PREFETCH     ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_L1D WRITE        ACCESS:    2946700 HIT:    2742386 MISS:     204314 MSHR_MERGE:      47067
cpu0->cpu0_L1D TRANSLATION  ACCESS:        302 HIT:         10 MISS:        292 MSHR_MERGE:          8
cpu0->cpu0_L1D PREFETCH REQUESTED:          0 ISSUED:          0 USEFUL:          0 USELESS:          0
cpu0->cpu0_L1D AVERAGE MISS LATENCY: 56.36 cycles
cpu0->cpu0_ITLB TOTAL        ACCESS:     966913 HIT:     966913 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_ITLB LOAD         ACCESS:     966913 HIT:     966913 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_ITLB RFO          ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_ITLB PREFETCH     ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_ITLB WRITE        ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_ITLB TRANSLATION  ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_ITLB PREFETCH REQUESTED:          0 ISSUED:          0 USEFUL:          0 USELESS:          0
cpu0->cpu0_ITLB AVERAGE MISS LATENCY: - cycles
cpu0->cpu0_DTLB TOTAL        ACCESS:   10975330 HIT:   10721088 MISS:     254242 MSHR_MERGE:      39822
cpu0->cpu0_DTLB LOAD         ACCESS:   10975330 HIT:   10721088 MISS:     254242 MSHR_MERGE:      39822
cpu0->cpu0_DTLB RFO          ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_DTLB PREFETCH     ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_DTLB WRITE        ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_DTLB TRANSLATION  ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->cpu0_DTLB PREFETCH REQUESTED:          0 ISSUED:          0 USEFUL:          0 USELESS:          0
cpu0->cpu0_DTLB AVERAGE MISS LATENCY: 5.558 cycles
cpu0->LLC TOTAL        ACCESS:     409084 HIT:     321335 MISS:      87749 MSHR_MERGE:          0
cpu0->LLC LOAD         ACCESS:     165001 HIT:     121941 MISS:      43060 MSHR_MERGE:          0
cpu0->LLC RFO          ACCESS:     115264 HIT:      77311 MISS:      37953 MSHR_MERGE:          0
cpu0->LLC PREFETCH     ACCESS:          0 HIT:          0 MISS:          0 MSHR_MERGE:          0
cpu0->LLC WRITE        ACCESS:     128567 HIT:     122082 MISS:       6485 MSHR_MERGE:          0
cpu0->LLC TRANSLATION  ACCESS:        252 HIT:          1 MISS:        251 MSHR_MERGE:          0
cpu0->LLC PREFETCH REQUESTED:          0 ISSUED:          0 USEFUL:          0 USELESS:          0
cpu0->LLC AVERAGE MISS LATENCY: 297.8 cycles

DRAM Statistics

Channel 0 RQ ROW_BUFFER_HIT:       3180
  ROW_BUFFER_MISS:      77820
  AVG DBUS CONGESTED CYCLE: 11.93
Channel 0 WQ ROW_BUFFER_HIT:      12400
  ROW_BUFFER_MISS:      46522
  FULL:          0
Channel 0 REFRESHES ISSUED:       2560
```

$$\text{Miss Rate} = \frac{\#\text{ of misses}}{\#\text{ of total accesses}} \times 100 = \frac{87749}{409084} \times 100 = \boxed{21.45\%}$$

$$\text{Hit Rate} = \frac{\#\text{ of hits}}{\#\text{ of total accesses}} \times 100 = \frac{321335}{409084} \times 100 = \boxed{78.55\%}$$

$$\text{Cumulative IPC} = \boxed{1.302}$$

همان طور که دیده می شود این سیاست از MRU به مراتب نتایج قابل قبول تری داشته اما در مقایسه با دو سیاست LRU و LFU بسیار به آن ها شباهت دارد.

۵ مقایسهٔ عملکردها

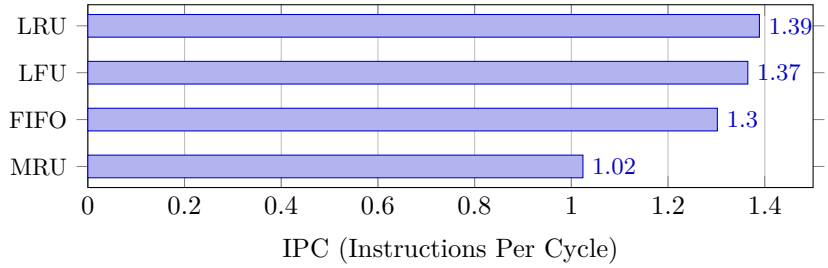
تمامی شبیه سازی های انجام شده روی لایهٔ LLC یعنی آخرین لایهٔ کش پیش از DRAM انجام شده و شرایط تست به جز Replacement Policy ها یکسان بوده است.

مشخصات لایه های مختلف کش استفاده شده:

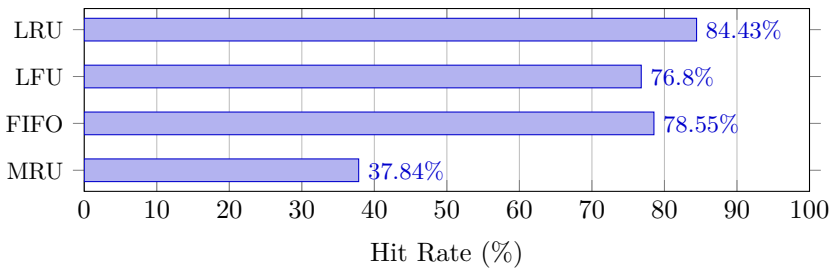
Table 1: Cache Hierarchy Parameters

Cache Level	Size	Associativity	Latency
L1I	32 KB	8-way	4 cycles
L1D	48 KB	12-way	5 cycles
L2	512 KB	8-way	10 cycles
LLC	2 MB	16-way	20 cycles

مقایسهٔ پارامتر Cumulative IPC



مقایسهٔ پارامتر Hit Rate



می توان علت اصلی کارایی مطلوب LRU را به این دلیل دانست که از اصول Locality Spatial و Locality Temporal به خوبی بهره می گیرد و در عین حال پیچیدگی زیادی ندارد. البته شایان ذکر است که روش هایی مانند شبه LRU وجود دارند

که پیچیدگی کمتری دارد. می توان ضعف LFU را در پیچیده بودن آن دانست. MRU هرچند ممکن است برای چینش های خاصی از Instruction ها مناسب باشد اما همان طور که در این شبیه سازی ها دیدیم، بدترین کارایی مربوط به MRU می شود.

۶ بخش امتیازی: پیاده سازی سیاست به وسیلهٔ پیش بینی تایم باقی مانده تا اخراج هر بلاک Mockingjay

در این سیاست سه جزء اساسی وجود دارد:

- Sampled Cache
- RDP = Reuse Distance Predictor
- ETR = Estimate Time Remaining

روند کلی به این صورت است که ابتدا یک مجموعه از ست ها را که به صورت عادلانه در LLC پخش شده اند را انتخاب کرده و به عنوان سمپل های خود در نظر می گیریم و در Sampled Cache قرار می دهیم. در این ساختمان داده همچنین آخرین کلاکی که در آن درخواست دسترسی به آن بلاک را دریافت کردیم و نیز به همراه سایر اطلاعات مورد نیاز که در ادامه آن ها را شرح خواهیم داد در ساختمان داده ذخیره می کنیم.

پس از این با گرفتن هر Hit جدید فاصلهٔ بین کلاک قبلی که بلاک را دیده ایم تا الان محاسبه می کنیم و نتیجه را به همراه آدرس بلاک در اختیار RDP قرار می دهیم.

حال ماژول RDP وظیفه دارد تا با اطلاعاتی که از قبل پیش بینی کرده و فاصلهٔ جدیدی که دریافت کرده، پیش بینی قبلی اش را مورد ارزیابی قرار دهد.

برای این کار باید به outlier ها توجه کنیم تا پیش بینی ما دچار خطای زیادی نشود و مقدار تغییرات را حساب شده در پیش بینی خود تاثیر دهیم.

در نهایت هر گاه بلاک جدیدی وارد کش می شود مقدار اولیهٔ ETR خود را از RDP دریافت کرده و در هر کلاک از تمامی ETR های داخل کش یک واحد کم می شود. در نهایت در موقع اخراج بلاکی باید اخراج شود که بیشترین مقدار قدر مطلقی را دارد. چرا که هر چه این مقدار بیشتر باشد یعنی تا فراخوانی مجدد آن بلاک فاصلهٔ بیشتری داریم.

همچنین شایان ذکر است اگر با دو مقدار مساوی ماکزیمم مواجه شدیم و مقدار اصلی آنها یکی منفی و دیگری مثبت بود، باید آنی را که منفی است اخراج کنیم چرا که در واقع overdue شده است.

حال با جزئیات به این سه ماژول و پیاده سازی آن ها می پردازیم:

۱.۶ Sampled Cache

ابتدا باید مشخص کنیم که چه ست هایی را قرار است به عنوان نمونه یاد بگیریم. در مجموع فرض کنید قرار است ۳۲ نمونه ست را یاد بگیریم. پس بهتر است ست های ۰ و ۳۲ و ۶۴ و ... را انتخاب کنیم.

پس از این باید ببینیم ست فراخوانی شدهٔ جدید آیا در لیست سمپل های ما قرار دارد یا خیر. اگر قرار ندارد که کاری نیاز نداریم انجام بدهیم.

اما اگر قرار داشته باشد، باید حال در همین کش sample شدهٔ خودمان آیا از پیش قرار داشته یا خیر.

اگر از پیش قرار داشته، کافی است فاصله اش را از فرمول زیر حساب کرده:

$$\text{reused distance} = \text{current time - last time}$$

و سپس این فاصله را در اختیار RDP گذاشته تا بر اساس آن پیش بینی اش را دقیق تر نماید. در ادامه باید last time و شمارهٔ PC این entry را آپدیت کنیم.

حال اگر از قبل در کش Sampled این entry وجود نداشت یعنی با miss مواجه هستیم و حالا باید یکی از مقادیر موجود در کش را اخراج کنیم. در این مرحله از همان سیاست LRU استفاده کرده و قربانی را اخراج می کنیم. باید این را ذکر کنیم که این قربانی را می توان SCAN دانست یعنی تنها یک بار مورد استفاده قرار گرفته و در ادامه دیگر به آن دسترسی انجام نشده به همین دلیل فاصلهٔ آن را ∞قرار می دهیم.

می توان این توضیحات فارسی را به شکل یک الگوریتم این گونه نشان داد:

Algorithm 1 Sample Cache

Require: Memory address, Program Counter (PC)

```
1: function SCA(address, PC)
2:     set_index ← (address ÷ # of blocks) mod # of sets
3:     x ← total number of sets
4:     y ← number of samples
5:     sampleList ← [0, x/y, 2x/y, 3x/y, 4x/y, ...]   [5 ways]
6:     if set_index ∈ sampleList then
7:         tag ← setTag(address)
8:         entry ← SampleCache.Find(set_index, tag)
9:         if entry then
10:            lastTime ← entry_timestamp
11:            PC ← entry.PC
12:            reused_distance ← currentTime − lastTime
13:            RDP_train(PC, reused_distance)
14:            entry_timestamp ← current_time
15:            entry.PC ← PC
16:        else
17:            victim ← SampleCache.Find_victim(set_index)
18:            victim_PC ← victim.PC
19:            RDP_train(victim_PC, ∞)                ▷ This is a scan
20:            victim_valid_bit ← 1
21:            victim_tag ← tag
22:            victim.PC ← PC
23:            victim_last_Time ← current_Time
```

حال بر اساس همین الگوریتم که بر آمده از مقالهٔ معرفی شده است، کد C را می توانیم این گونه بازنویسی کنیم:

```
for (int i = 0; i < 32; ++i) {
        sampled_set_indices.push_back(i * (NUM_SET / 32));
    }
```

```
void SampledCache::handle_access(uint64_t full_addr, uint64_t
    pc_signature, int set_in_llc, int current_timestamp) {
    uint32_t internal_set = set_in_llc % NUM_SETS;
    uint64_t tag = full_addr >> 12;

    int hit_way = -1;
    for (int i = 0; i < NUM_WAYS; ++i) {
        if (sets[internal_set][i].valid && sets[internal_set][i].tag ==
            tag) {
            hit_way = i;
            break;
        }
    }

    if (hit_way != -1) {
        auto& entry = sets[internal_set][hit_way];
        int time_elapsed = (current_timestamp >= entry.timestamp) ? (
            current_timestamp - entry.timestamp) : (current_timestamp -
            entry.timestamp + 256);

        rdp.train(entry.pc_signature, time_elapsed);

        entry.pc_signature = pc_signature;
        entry.timestamp = current_timestamp;
    } else {
        int victim_way = -1;
        int max_age = -1;
        for (int i = 0; i < NUM_WAYS; ++i) {
            if (!sets[internal_set][i].valid) {
                victim_way = i;
                break;
            }
            int age = (current_timestamp >= sets[internal_set][i].
                timestamp) ? (current_timestamp - sets[internal_set][i].
                timestamp) : (current_timestamp - sets[internal_set][i].
                timestamp + 256);
```

```
            if (age > max_age) {
                max_age = age;
                victim_way = i;
            }
        }

        if (sets[internal_set][victim_way].valid) {
            rdp.train(sets[internal_set][victim_way].pc_signature, RDP::
                INF_RD);
        }

        sets[internal_set][victim_way] = {true, tag, pc_signature,
            current_timestamp};
    }
}
```

RDP ۲.۶

حال به سراغ RDP می رویم که شامل دو بخش است. predict و train

تابع predict صرفا با استفاده از بیت های پایینی PC مقادیر پیش بینی شده را بر می گرداند.

اما دربارهٔ train همان طور که گفته شد بر اساس اختلاف فاصلهٔ دو درخواست از یک entry یکسان عمل می کند و زمان درخواست بعدی را با الگوریتم سادهٔ زیر پیش بینی می کند:

Algorithm 2 RDP Train Algorithm

Require: Program Counter (PC), observed distance

function RDP_Train(PC, observed_distance)
    entry ← RDP.Find(PC)
    old_Prediction ← entry.predicted_distance
    diff ← observed_distance − old_Prediction
    nudge ← min $\left(1, \frac{|\text{diff}|}{16}\right)$
    if observed_distance > old_prediction **then**
        entry.predicted_distance ← entry.predicted_distance + nudge
    else
        entry.predicted_distance ← entry.predicted_distance − nudge

حال به راحتی می توان این شبه کد را به زبان C تبدیل کرد:

```
int RDP::temporal_difference(int init, int sample) {
```

```
    if (sample > init) {
        int diff = (sample - init) / 16;
        return min(init + max(1, diff), INF_RD);
    }
    if (sample < init) {
        int diff = (init - sample) / 16;
        return max(init - max(1, diff), 0);
    }
    return init;
}
```

```
void RDP::train(uint64_t pc_signature, int sample) {
    uint32_t index = pc_signature % PREDICTOR_SIZE;
    rdp_table[index] = temporal_difference(rdp_table[index], sample);
}
```

ETR ۳.۶

با توجه به توضیحات داده شده به راحتی می توان الگوریتم این بخش را نیز به شکل زیر نوشت:

Algorithm 3 ETR-Based Victim Selection

Require: triggering_cpu, instr_id, set, current_set, ip, full_addr, type

Ensure: Victim way index or NUM_WAY if no victim found

```
1: function FIND_VICTIM(triggering_cpu, instr_id, set, current_set, ip, full_addr, type)
2:     for i ← 0 to NUM_WAY − 1 do
3:         if current_set[i].valid = false then
4:             return i                                   ▷ Return first invalid way

5:     pc_signature ← get_pc_signature(ip, false)
6:     predicted_rd ← rdp.predict(pc_signature)
7:     if type ≠ WRITE ∧ (predicted_rd > MAX_RD_THRESHOLD) then
8:         return NUM_WAY                                 ▷ No victim found case

9:     victim_way ← 0
10:    max_abs_etr ← −1
11:    for i ← 0 to NUM_WAY − 1 do
12:        current_etr ← etr_counters[set][i]
13:        current_abs_etr ← |current_etr|

14:        if current_abs_etr > max_abs_etr then
15:            max_abs_etr ← current_abs_etr
16:            victim_way ← i
17:        else if current_abs_etr = max_abs_etr then
18:            if etr_counters[set][victim_way] ≥ 0 ∧ current_etr < 0 then
19:                victim_way ← i                         ▷ Prefer negative ETR when tied
20:    return victim_way
```

و کد C متناظر به این شکل است:

```
long myMOCKINGJAY::find_victim(...) {
    for (long i = 0; i < NUM_WAY; ++i) {
        if (!current_set[i].valid) {
            return i;
        }
    }

    uint64_t pc_signature = get_pc_signature(ip.to<uint64_t>(), false);
    int predicted_rd = rdp.predict(pc_signature);
    if (type != access_type::WRITE && (predicted_rd > MAX_RD_THRESHOLD))
        {
        return NUM_WAY;
    }

    long victim_way = 0;
    int max_abs_etr = -1;
    for (long i = 0; i < NUM_WAY; ++i) {
        int current_etr = etr_counters[set][i];
        int current_abs_etr = abs(current_etr);

        if (current_abs_etr > max_abs_etr) {
            max_abs_etr = current_abs_etr;
            victim_way = i;
        } else if (current_abs_etr == max_abs_etr) {
            if (etr_counters[set][victim_way] >= 0 && current_etr < 0) {
                victim_way = i;
            }
        }
    }
    return victim_way;
}
```

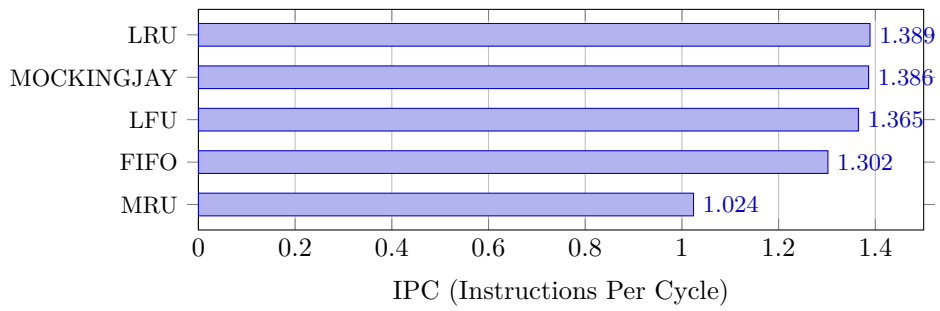
در آخر نیز باید اشاره کنیم با هر hit باید یک عدد از ETR ها کم شود به شرط آنکه آن entry یک SCAN نباشد. همچنین باید چک کنیم که آیا فراخوانی اخیر جزوی از sample های ما است یا خیر و اگر هست باید آن را در اختیار Sampled Cache نیز قرار بدهیم:

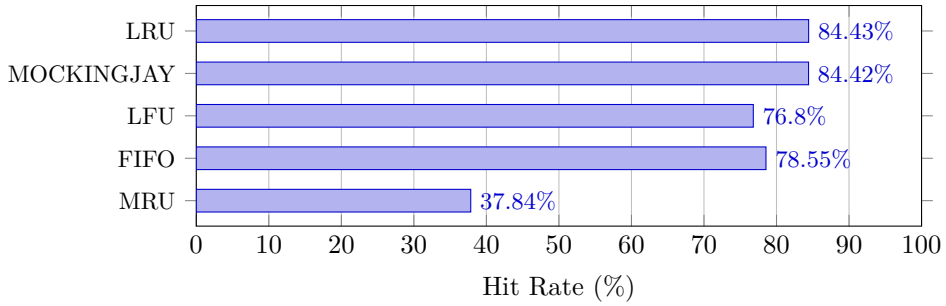
```
void myMOCKINGJAY::update_replacement_state(uint32_t triggering_cpu, long
    set, long way, champsim::address full_addr, champsim::address ip,
                                champsim::address victim_addr,
                                    access_type type, uint8_t hit) {
```

```
    if (way >= NUM_WAY) return;
    if (type == access_type::WRITE) return;

    uint64_t pc_signature = get_pc_signature(ip.to<uint64_t>(), hit);

    if (hit) {
        int predicted_rd = rdp.predict(pc_signature);
        etr_counters[set][way] = (predicted_rd > MAX_RD_THRESHOLD) ?
            INF_ETR : predicted_rd / GRANULARITY;
    }

    etr_clock[set]++;
    if (etr_clock[set] >= GRANULARITY) {
        etr_clock[set] = 0;
        for (int i = 0; i < NUM_WAY; ++i) {
            if (abs(etr_counters[set][i]) < INF_ETR) {
                etr_counters[set][i]--;
            }
        }
    }

    if (is_sampled_set(set)) {
        set_timestamps[set] = (set_timestamps[set] + 1) % 256;
        sampled_cache.handle_access(full_addr.to<uint64_t>(),
            pc_signature, set, set_timestamps[set]);
    }
}
```

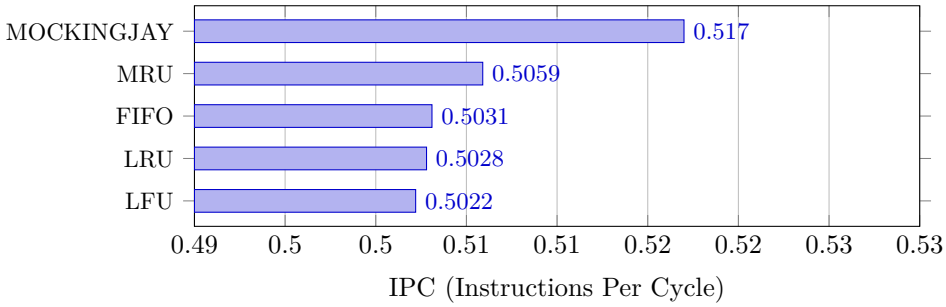
۴.۶ مقایسهٔ عملکرد

با مبنا قرار دادن همان دستور قبلی با ۶۰ میلیون دستور warmup و ۴۰ میلیون دستور شبیه سازی و با همان trace گذشته، مقادیر IPC و Hit Rate را مقایسه می کنیم:





اما خالی از لطف نیست تا این سیاست ها را روی یک trace دیگر مانند lbm_۱۰۴B که ساختار پیچیده ای دارد نیز انجام دهیم:



در کل می توان گفت در توزیع های مختلف داده عملکرد مشابهی با LRU داشته اما زمانی که الگوی موجود در داده پیچیده باشد مانند lbm، عملکرد بهتری در مقایسه با دیگر سیاست ها نشان می دهد.

از دیدگاه پیچیدگی می توان گفت در میان این ۵ سیاست، از همه پیچیدگی بیشتری داشته و حافظهٔ بیشتری نسبت به آنها اشغال می کند. چرا که دارای ۳ بخش است که هر یک باید داده هایی را ذخیره کنند.

اما با این حال به دلیل آن که پیش بینی زمان انجام فراخوانی ها بسیار کمک کننده است، نسبت به سایر سیاست ها کارایی نامطلوبی نداشته و در تست های مختلف با توزیع داده های گوناگون کارایی مناسبی دارد.