

[پروژه پایان ترم معماری کامپیوتر، شبیه‌سازی cache با ChampSim]

| نازنین فروتن، ۴۰۲۲۴۳۰۸۴

| ریحانه داداش‌پور، ۴۰۲۲۴۳۰۶۰

| استاد چشمی‌خانی

| ترم بهار ۴۰۴

[بخش اول پروژه]

* هدف اصلی حافظه نهان (cache) کم کردن زمان دسترسی به حافظه اصلی یا همون RAM هست.

سیاست‌های جایگزینی یا Replacement Policies

- وقتی کش پر می‌شود، نیاز به جایگزینی یک بلوک جدید با یکی از بلوک‌های قدیمی داریم. هر کدام از سیاست‌های جایگزینی، یک رویکرد متفاوت در انتخاب بلوکی که باید حذف شود را در بر می‌گیرند.

(Least Recently Used) LRU - 1

در این سیاست، قدیمی‌ترین خط مورد استفاده حذف می‌شود. منطق این سیاست آن است که بلوک داده‌ای که طولانی‌مدت استفاده نشده، احتمالاً در آینده نزدیک هم استفاده نخواهد شد و می‌توان حذفش کرد. برای پیاده‌سازی LRU، نیاز به زمان‌سنج داریم که آخرین زمان استفاده از هر کدام از بلوک‌ها را ثبت کند که براساس آن بلوک با بیشترین زمان حذف شود.

(Least Frequently Used) LFU - 2

در این سیاست، بلوکی که کمترین دفعات استفاده را داشته باشد، حذف می‌شود. منطق این سیاست آن است که بلوک داده‌ای که کمتر از بقیه استفاده شده، احتمال کمتری دارد که در آینده هم مورد استفاده قرار بگیرد. برای پیاده‌سازی LFU، هر بلوک از کش نیازمند شمارنده‌ی فرکانس است که تعداد دفعات دسترسی به آن بلوک را ثبت کند.

(Most Recently Used) MRU - 3

این سیاست همان طور که از نامش پیداست، دقیقاً برعکس LRU عمل می‌کند و تازه‌ترین بلوک مورد استفاده را حذف می‌کند. همچنین این الگوریتم نسبت به LRU کارایی کمتری دارد و در مواردی خاص مثل نیاز به دسترسی چرخشی به داده‌ها مورد استفاده قرار می‌گیرد.

(First In First Out FIFO - 4): بلوکی که زودتر وارد حافظه می‌شود، زودتر هم حذف می‌شود. ساده‌ترین سیاست جایگزینی محسوب می‌شود که در آن بلوکی که زودتر از همه وارد کش شده حذف می‌شود. (فقط ترتیب ورود داده‌ها در نظر گرفته می‌شود). معمولاً در سیستم‌های ساده‌تر مورد استفاده قرار می‌گیرد (به‌حاطر سادگی در پیاده‌سازی) ولی ممکن است که بلوک‌هایی که مداوم از آن‌ها استفاده می‌شود را هم حذف کند.

* از بین فایل‌های موجود، از `astar_163B.trace.xz` استفاده می‌کنیم که در پوشه `traces` قرار دادیم.

```
rein@DESKTOP-C4TMSM1:~/ChampSim/replacement$ ls
drrip lru random ship srrip
```

پوشه اصلی replacement
داخل پوشه کلون شده ChampSim
همست که در ابتدا شامل فایل‌های زیر می‌باشد:

| پیاده‌سازی سیاست‌های جایگزینی:

الف) پیاده‌سازی الگوریتم LRU:

بین سیاست‌هایی که ما باید بررسی کنیم اینجا فقط lru به‌طور پیشفرض تعریف شده که براساسش بقیه سیاست‌ها رو پیاده می‌کنیم.

```
rein@DESKTOP-C4TMSM1:~/ChampSim/replacement$ cd lru
rein@DESKTOP-C4TMSM1:~/ChampSim/replacement/lru$ ls
lru.cc  lru.h
```

شامل فایل‌های lru.cc که پیاده‌سازی اصلی الگوریتم lru و فایل lru.h که تعریف این الگوریتم به‌عنوان ماثول برای خود champsim هست.

```
#include "lru.h"
#include <algorithm>
#include <cassert>

lru::lru(CACHE* cache) : lru(cache, cache->NUM_SET, cache->NUM_WAY) {}

lru::lru(CACHE* cache, long sets, long ways) : replacement(cache), NUM_WAY(ways),
last_used_cycles(static_cast<std::size_t>(sets * ways), 0) {}

long lru::find_victim(uint32_t triggering_cpu, uint64_t instr_id, long set, const
champsim::cache_block* current_set, champsim::address ip,
champsim::address full_addr, access_type type) {
    auto begin = std::next(std::begin(last_used_cycles), set * NUM_WAY);
    auto end = std::next(begin, NUM_WAY);

    auto victim = std::min_element(begin, end);
    assert(begin <= victim);
    assert(victim < end);
    return std::distance(begin, victim);
}

void lru::replacement_cache_fill(uint32_t triggering_cpu, long set, long way, champsim::address
full_addr, champsim::address ip, champsim::address victim_addr,
access_type type) {
    last_used_cycles.at((std::size_t)(set * NUM_WAY + way)) = cycle++;
}

void lru::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 (hit && access_type{type} != access_type::WRITE)
        last_used_cycles.at((std::size_t)(set * NUM_WAY + way)) = cycle++;
}
```

```

#ifndef REPLACEMENT_LRU_H
#define REPLACEMENT_LRU_H

#include <vector>
#include "cache.h"
#include "modules.h"

class lru : public champsim::modules::replacement
{
    long NUM_WAY;
    std::vector<uint64_t> last_used_cycles;
    uint64_t cycle = 0;

public:
    explicit lru(CACHE* cache);
    lru(CACHE* cache, long sets, long ways);

    long find_victim(uint32_t triggering_cpu, uint64_t instr_id, long set, const champsim::cache_block* current_set, champsim::address ip,
                     champsim::address full_addr, access_type type);
    void replacement_cache_fill(uint32_t triggering_cpu, long set, long way, champsim::address full_addr,
                                champsim::address ip, champsim::address victim_addr,
                                access_type type);
    void 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);
};

#endif

```

| پارامترهای خواسته شده:

:IPC (Instruction Per Cycle) -1

CPU 0 cumulative IPC: 0.9749 instructions: 40000002 cycles: 41028255

یعنی در هر سیکل به طور میانگین ۹۷۴۹ دستور اجرا شده است.

اینجا مقدار IPC نسبتاً پایین است که یعنی در شبیه‌سازی دستور کمی به‌ازای هر سیکل پردازش شده که می‌تواند به‌خاطر تعداد miss rate بالا در سطوح کش بالاتر باشد.

در لایه‌های Hit Rate و Miss Rate -۲

یا کش لایه آخر): Last Level Cache LLC -

Access: 140359

Hit: 107235

Miss: 33124

$$* \text{ Hit Rate: } \frac{107235}{140359} \approx 76.4\%$$

$$* \text{ Miss Rate: } \frac{33124}{140359} \approx 23.6\%$$

باتوجه به نتایج، حدود ۳۳ درصد است که عدد بالایی محسوب می‌شود. (یعنی بخش زیادی از دسترسی‌ها باید به حافظه اصلی ارسال شوند).

:L1D -

Access:	11302446
Hit:	11070278
Miss:	232168

$$* \text{ Hit Rate: } \frac{11070278}{11302446} \approx 97.9\%$$

$$* \text{ Miss Rate: } \frac{33124}{140359} \approx 2.1\%$$

در L1D miss rate خیلی پایین و نشان‌دهنده عملکرد خیلی خوب این لایه است. یعنی بیشتر داده‌ها در این لایه ذخیره شده و خیلی کم نیاز به لایه‌های بالاتر کش پیدا می‌شود.

:L2C -

Access:	230524
Hit:	150279
Miss:	80245

$$* \text{ Hit Rate: } \frac{150279}{230524} \approx 65.2\%$$

$$* \text{ Miss Rate: } \frac{80245}{230524} \approx 34.8\%$$

در این لایه از کش یعنی L2C خیلی بالاست. (که یعنی داده‌ها در این لایه به سختی پیدا می‌شوند).

LLC Average Miss Latency - ۳

برابر با: 165.2 cycles

--

* مقایسه بین عملکرد سیاست‌ها در لایه LLC بررسی می‌شود و بقیه لایه‌ها در بقیه الگوریتم‌ها با lru یکسان هستند.

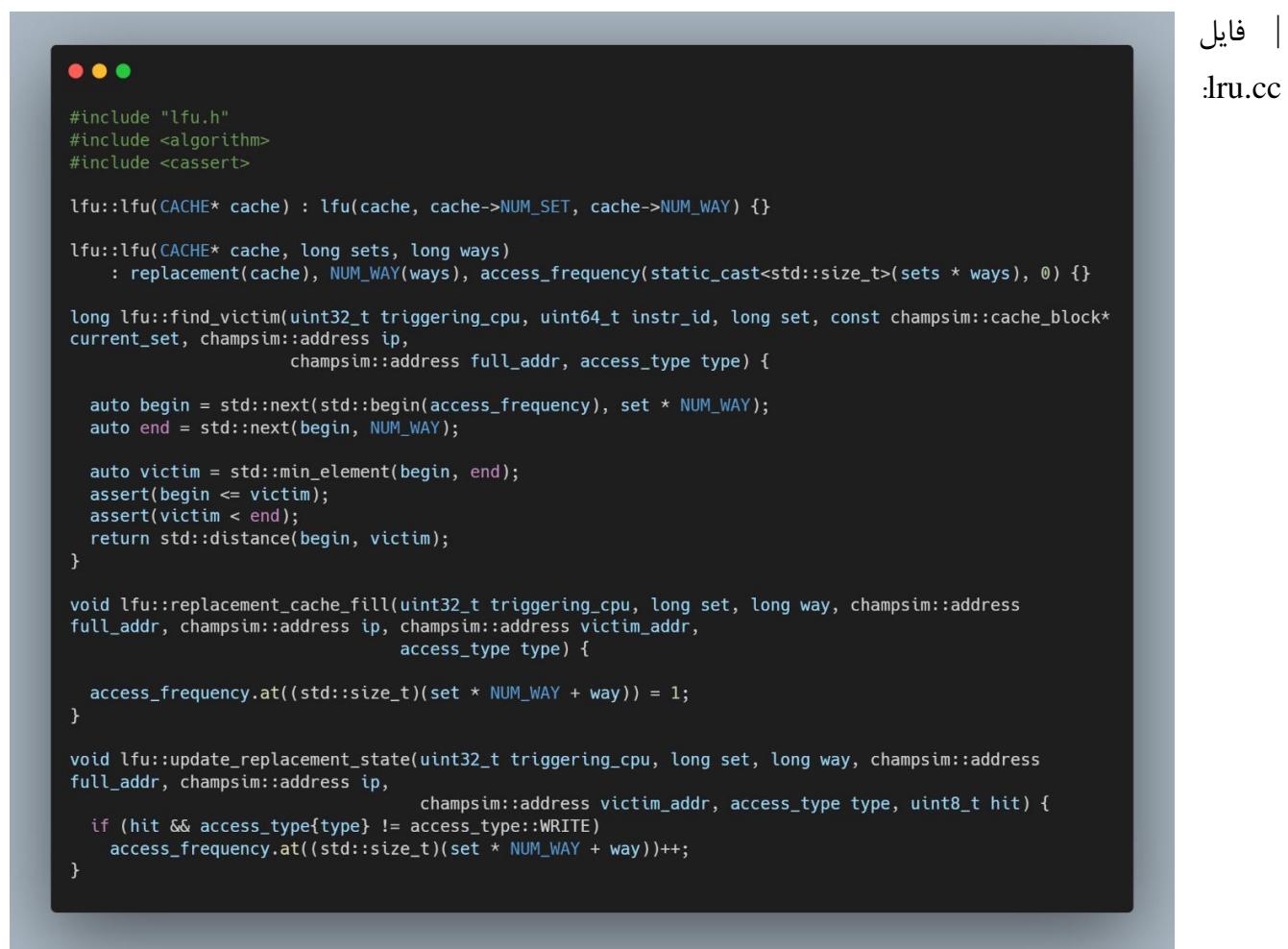
* برای تست کردن هر کدام از الگوریتم‌ها باید اسم policy موردنظر خود را در گزینه replacement بخش champsim_config.json مشخص کنیم. (به صورت پیشفرض روی سیاست lru تنظیم شده، ولی با تغییر خط مربوط به کش موردنظر می‌شه تغییرش داد).

* بعد از ذخیره کدها و تنظیم پیکربندی (*champsim_config*) با وارد کردن دستور *make clean* و سپس *make* کد مربوطه کامپایل می‌شود و سپس برای شبیه‌سازی کش، از دستور زیر استفاده می‌کنیم:

./bin/champsim --warmup-instructions 60000000 --simulation-instructions 400000000
~/traces/astar_163B.trace.xz

ب) پیاده‌سازی الگوریتم LFU

برای هر بلوک کش یه counter تعریف می‌کنیم که هر بار که hit شود، به آن یکی اضافه می‌گردد. در انتهای هنگام replacement، بلوکی که مقدار counter آن از همه کمتر باشد حذف می‌شود. برای این کار، به جای بردار access_frequency که در الگوریتم LRU وجود دارد، از یک بردار به نام last_used_cycles استفاده می‌کنیم که تعداد دفعات استفاده از هر بلوک کش را ثبت می‌کند. هر داده‌ی جدیدی که به کش وارد می‌شود، مقدار این بردار برای آن برابر ۱ در نظر گرفته می‌شود. (با تابع replacement_cache_fill)



فایل | lru.cc

```
#include "lfu.h"
#include <algorithm>
#include <cassert>

lfu::lfu(CACHE* cache) : lfu(cache, cache->NUM_SET, cache->NUM_WAY) {}

lfu::lfu(CACHE* cache, long sets, long ways)
    : replacement(cache), NUM_WAY(ways), access_frequency(static_cast<std::size_t>(sets * ways), 0) {}

long lfu::find_victim(uint32_t triggering_cpu, uint64_t instr_id, long set, const champsim::cache_block* current_set, champsim::address ip,
                      champsim::address full_addr, access_type type) {
    auto begin = std::next(std::begin(access_frequency), set * NUM_WAY);
    auto end = std::next(begin, NUM_WAY);

    auto victim = std::min_element(begin, end);
    assert(begin <= victim);
    assert(victim < end);
    return std::distance(begin, victim);
}

void lfu::replacement_cache_fill(uint32_t triggering_cpu, long set, long way, champsim::address full_addr, champsim::address ip, champsim::address victim_addr,
                                 access_type type) {
    access_frequency[(std::size_t)(set * NUM_WAY + way)] = 1;
}

void lfu::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 (hit && access_type{type} != access_type::WRITE)
        access_frequency[(std::size_t)(set * NUM_WAY + way)]++;
}
```

| فایل lfu.h نیز به این صورت است:

```
#ifndef REPLACEMENT_LFU_H
#define REPLACEMENT_LFU_H

#include <vector>
#include "cache.h"
#include "modules.h"

class lfu : public champsim::modules::replacement
{
    long NUM_WAY;
    std::vector<uint64_t> access_frequency;
    uint64_t counter = 0;

public:
    explicit lfu(CACHE* cache);
    lfu(CACHE* cache, long sets, long ways);

    long find_victim(uint32_t triggering_cpu, uint64_t instr_id, long set, const champsim::cache_block* current_set, champsim::address ip,
                     champsim::address full_addr, access_type type);
    void replacement_cache_fill(uint32_t triggering_cpu, long set, long way, champsim::address full_addr,
                                champsim::address ip, champsim::address victim_addr,
                                access_type type);
    void 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);
};

#endif
```

ساختار کلی کد:

- در فایل lfu.h که کلاس lfu از کلاس replacement ارث بری می‌کند. مثل lru، آدرس هر بلوک توى access_frequency با فرمول (set * NUM_WAY + way) معین می‌شود.
- تابع update_replacement_state: اگر داده نامبرده hit شد و همچنین عملیات در حال انجام، عملیات نوشتن نبود، به counter بلوک موردنظر یکی اضافه می‌شود.
- تابع :find_victim در مرحله نهایی سیاست جایگزینی، داخل ست پر شده بلوکی رو پیدا می‌کند که کمترین مقدار counter را داشته باشد.

نتایج به دست آمده در ترمینال بعد از اتمام شبیه سازی:

```
rein@DESKTOP-C4TMSM1:~/ChampSim$ ./bin/champsim --warmup-instructions 60000000 --simulation-instructions 40000000 ~/traces/astar_163B.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: 10000002 cycles: 3327695 heartbeat IPC: 3.005 cumulative IPC: 3.005 (Simulation time: 00 hr 00 min 59 sec)
Heartbeat CPU 0 instructions: 20000004 cycles: 6068533 heartbeat IPC: 3.649 cumulative IPC: 3.296 (Simulation time: 00 hr 01 min 50 sec)
Heartbeat CPU 0 instructions: 30000006 cycles: 8630261 heartbeat IPC: 3.904 cumulative IPC: 3.476 (Simulation time: 00 hr 02 min 37 sec)
Heartbeat CPU 0 instructions: 40000006 cycles: 11149894 heartbeat IPC: 3.969 cumulative IPC: 3.587 (Simulation time: 00 hr 03 min 30 sec)
Heartbeat CPU 0 instructions: 50000010 cycles: 13650384 heartbeat IPC: 3.999 cumulative IPC: 3.663 (Simulation time: 00 hr 04 min 17 sec)
Warmup finished CPU 0 instructions: 60000000 cycles: 16154443 cumulative IPC: 3.714 (Simulation time: 00 hr 05 min 03 sec)
Warmup complete CPU 0 instructions: 60000000 cycles: 16154443 cumulative IPC: 3.714 (Simulation time: 00 hr 05 min 03 sec)
Heartbeat CPU 0 instructions: 60000012 cycles: 16154446 heartbeat IPC: 3.994 cumulative IPC: 4 (Simulation time: 00 hr 05 min 03 sec)
Heartbeat CPU 0 instructions: 70000016 cycles: 26866083 heartbeat IPC: 0.9336 cumulative IPC: 0.9336 (Simulation time: 00 hr 06 min 36 sec)
Heartbeat CPU 0 instructions: 80000018 cycles: 38195706 heartbeat IPC: 0.8826 cumulative IPC: 0.9074 (Simulation time: 00 hr 08 min 06 sec)
Heartbeat CPU 0 instructions: 90000018 cycles: 49222004 heartbeat IPC: 0.9069 cumulative IPC: 0.9072 (Simulation time: 00 hr 09 min 32 sec)
Simulation finished CPU 0 instructions: 40000002 cycles: 44127143 cumulative IPC: 0.9065 (Simulation time: 00 hr 11 min 01 sec)
Simulation complete CPU 0 instructions: 40000002 cycles: 44127143 cumulative IPC: 0.9065 (Simulation time: 00 hr 11 min 01 sec)

ChampSim completed all CPUs

== Simulation ==
CPU 0 runs /home/rein/traces/astar_163B.trace.xz

Region of Interest Statistics

CPU 0 cumulative IPC: 0.9065 instructions: 40000002 cycles: 44127143
CPU 0 Branch Prediction Accuracy: 77.23% MPKI: 33.35 Average ROB Occupancy at Mispredict: 14.85
Branch type MPKI
BRANCH_DIRECT_JUMP: 0
BRANCH_INDIRECT: 0
BRANCH_CONDITIONAL: 33.35
BRANCH_DIRECT_CALL: 0
BRANCH_INDIRECT_CALL: 0
BRANCH_RETURN: 0
```

```
cpu0->LLC TOTAL ACCESS: 140360 HIT: 35556 MISS: 104804 MSHR_MERGE: 0
cpu0->LLC LOAD ACCESS: 47305 HIT: 8525 MISS: 38780 MSHR_MERGE: 0
cpu0->LLC RFO ACCESS: 28794 HIT: 8039 MISS: 20755 MSHR_MERGE: 0
cpu0->LLC PREFETCH ACCESS: 0 HIT: 0 MISS: 0 MSHR_MERGE: 0
cpu0->LLC WRITE ACCESS: 60115 HIT: 17695 MISS: 42420 MSHR_MERGE: 0
cpu0->LLC TRANSLATION ACCESS: 4146 HIT: 1297 MISS: 2849 MSHR_MERGE: 0
cpu0->LLC PREFETCH REQUESTED: 0 ISSUED: 0 USEFUL: 0 USELESS: 0
cpu0->LLC AVERAGE MISS LATENCY: 105.3 cycles

DRAM Statistics

Channel 0 RQ ROW_BUFFER_HIT: 1346
ROW_BUFFER_MISS: 61037
AVG DBUS CONGESTED CYCLE: 19.09
Channel 0 WQ ROW_BUFFER_HIT: 5982
ROW_BUFFER_MISS: 37113
FULL: 0
Channel 0 REFRESHES ISSUED: 3677
rein@DESKTOP-C4TMSM1:~/ChampSim$
```

پارامترهای خواسته شده:

:IPC (Instruction Per Cycle) - 1

```
CPU 0 cumulative IPC: 0.9065 instructions: 40000002 cycles: 44127143
```

یعنی در هر سیکل به طور میانگین ۰،۹۰۶۵ دستور اجرا شده است.

از LRU کمتر است. ممکن است به خاطر miss rate بالا در لایه های میانی یا بالایی کش باشد که باعث می شود دستورات بیشتر به حافظه اصلی بروند و کارایی کاهش پیدا کند.

:(Last Level Cache یا) LLC در Hit Rate و Miss Rate - 2

Access: 140360

Hit: 35556

Miss: 104804

$$* \text{ Hit Rate: } \frac{35556}{140360} \approx 25.3\%$$

$$* \text{ Miss Rate: } \frac{104804}{140360} \approx 74.7\%$$

در این سیاست، مقدار hit rate خیلی کم است که یعنی مقدار زیادی از داده‌ها به کش دسترسی پیدا نکرده‌اند.

miss rate - نیاز به مدیریت داده‌های بیشتری دارد، این الگوریتم نمی‌تواند به درستی داده‌های جدیدتر یا کاماستفاده‌تر را مدیریت کند).

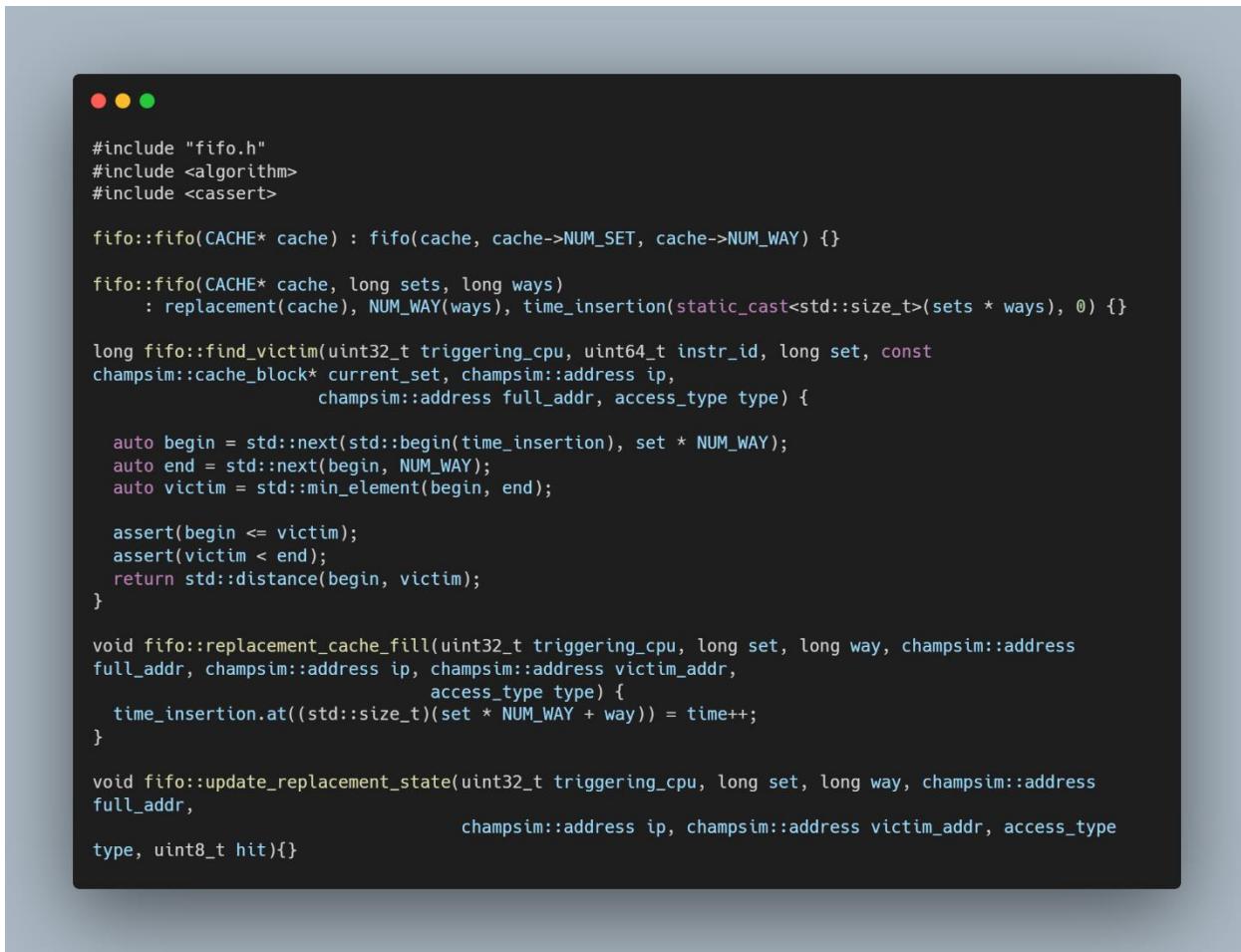
Average Miss Latency - 3

برابر با: 105.3 cycles (بهای هر miss، این مقدار طول می‌کشد تا داده‌ها از حافظه اصلی بارگذاری شوند).

ج) پیاده‌سازی الگوریتم FIFO

هر کدام از بلوک‌های موجود در cache، یک زمان ورود یا همان insertion time دارد که در زمان جایگزینی، قدیمی‌ترین بلوک (یعنی بلوکی که کمترین insertion time را دارد) حذف می‌شود.

فایل fifo.cc این الگوریتم:



```
#include "fifo.h"
#include <algorithm>
#include <cassert>

fifo::fifo(CACHE* cache) : fifo(cache, cache->NUM_SET, cache->NUM_WAY) {}

fifo::fifo(CACHE* cache, long sets, long ways)
    : replacement(cache), NUM_WAY(ways), time_insertion(static_cast<std::size_t>(sets * ways), 0) {}

long fifo::find_victim(uint32_t triggering_cpu, uint64_t instr_id, long set, const
champsim::cache_block* current_set, champsim::address ip,
                        champsim::address full_addr, access_type type) {

    auto begin = std::next(std::begin(time_insertion), set * NUM_WAY);
    auto end = std::next(begin, NUM_WAY);
    auto victim = std::min_element(begin, end);

    assert(begin <= victim);
    assert(victim < end);
    return std::distance(begin, victim);
}

void fifo::replacement_cache_fill(uint32_t triggering_cpu, long set, long way, champsim::address
full_addr, champsim::address ip, champsim::address victim_addr,
                                access_type type) {
    time_insertion.at((std::size_t)(set * NUM_WAY + way)) = time++;
}

void fifo::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){}
```

فایل fifo.h نیز به این صورت است:

```

● ● ●

#ifndef REPLACEMENT_FIFO_H
#define REPLACEMENT_FIFO_H

#include <vector>
#include "cache.h"
#include "modules.h"

class fifo : public champsim::modules::replacement
{
    long NUM_WAY;
    std::vector<uint64_t> time_insertion;
    uint64_t time = 0;

public:
    explicit fifo(CACHE* cache);
    fifo(CACHE* cache, long sets, long ways);

    long find_victim(uint32_t triggering_cpu, uint64_t instr_id, long set, const champsim::cache_block* current_set, champsim::address ip,
                     champsim::address full_addr, access_type type);
    void replacement_cache_fill(uint32_t triggering_cpu, long set, long way, champsim::address full_addr,
                                champsim::address ip, champsim::address victim_addr,
                                access_type type);
    void 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);
};

#endif

```

| نتایج به دست آمده در ترمینال بعد از اتمام شبیه سازی در سطح LLC (بقیه سطوح مشابه LRU):

```

rein@DESKTOP-C4TMSM1:~/ChampSim$ ./bin/champsim --warmup-instructions 60000000 --simulation-instructions 40000000 ~/traces/astar_163B.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: 10000002 cycles: 3327725 heartbeat IPC: 3.005 cumulative IPC: 3.005 (Simulation time: 00 hr 00 min 54 sec)
Heartbeat CPU 0 instructions: 20000004 cycles: 6068513 heartbeat IPC: 3.649 cumulative IPC: 3.296 (Simulation time: 00 hr 01 min 39 sec)
Heartbeat CPU 0 instructions: 30000008 cycles: 8630433 heartbeat IPC: 3.903 cumulative IPC: 3.476 (Simulation time: 00 hr 02 min 25 sec)
Heartbeat CPU 0 instructions: 40000011 cycles: 11150683 heartbeat IPC: 3.968 cumulative IPC: 3.587 (Simulation time: 00 hr 03 min 07 sec)
Heartbeat CPU 0 instructions: 50000011 cycles: 13651170 heartbeat IPC: 3.999 cumulative IPC: 3.663 (Simulation time: 00 hr 03 min 50 sec)
Heartbeat CPU 0 instructions: 60000000 cycles: 16155353 cumulative IPC: 3.714 (Simulation time: 00 hr 04 min 35 sec)
Warmup complete CPU 0 instructions: 60000000 cycles: 16155353 cumulative IPC: 3.714 (Simulation time: 00 hr 04 min 35 sec)
Heartbeat CPU 0 instructions: 60000012 cycles: 16155356 heartbeat IPC: 3.993 cumulative IPC: 4 (Simulation time: 00 hr 04 min 35 sec)
Heartbeat CPU 0 instructions: 70000012 cycles: 26244734 heartbeat IPC: 0.9911 cumulative IPC: 0.9911 (Simulation time: 00 hr 06 min 01 sec)
Heartbeat CPU 0 instructions: 80000013 cycles: 36598395 heartbeat IPC: 0.9658 cumulative IPC: 0.9783 (Simulation time: 00 hr 07 min 20 sec)
Heartbeat CPU 0 instructions: 90000013 cycles: 46682864 heartbeat IPC: 0.9916 cumulative IPC: 0.9827 (Simulation time: 00 hr 08 min 41 sec)
Simulation finished CPU 0 instructions: 40000002 cycles: 41050392 cumulative IPC: 0.9744 (Simulation time: 00 hr 09 min 56 sec)
Simulation complete CPU 0 instructions: 40000002 cycles: 41050392 cumulative IPC: 0.9744 (Simulation time: 00 hr 09 min 56 sec)

ChampSim completed all CPUs

== Simulation ==
CPU 0 runs /home/rein/traces/astar_163B.trace.xz

Region of Interest Statistics

CPU 0 cumulative IPC: 0.9744 instructions: 40000002 cycles: 41050392
CPU 0 Branch Prediction Accuracy: 77.23% MPKI: 33.35 Average ROB Occupancy at Mispredict: 14.37
Branch type MPKI
BRANCH_DIRECT_JUMP: 0
BRANCH_INDIRECT: 0
BRANCH_CONDITIONAL: 33.35
BRANCH_DIRECT_CALL: 0
BRANCH_INDIRECT_CALL: 0
BRANCH_RETURN: 0

```

```

cpu0->LLC TOTAL ACCESS: 140359 HIT: 104426 MISS: 35933 MSHR_MERGE: 0
cpu0->LLC LOAD ACCESS: 47305 HIT: 28839 MISS: 18466 MSHR_MERGE: 0
cpu0->LLC RFO ACCESS: 28794 HIT: 16545 MISS: 12249 MSHR_MERGE: 0
cpu0->LLC PREFETCH ACCESS: 0 HIT: 0 MISS: 0 MSHR_MERGE: 0
cpu0->LLC WRITE ACCESS: 60115 HIT: 57169 MISS: 2946 MSHR_MERGE: 0
cpu0->LLC TRANSLATION ACCESS: 4145 HIT: 1873 MISS: 2272 MSHR_MERGE: 0
cpu0->LLC PREFETCH REQUESTED: 0 ISSUED: 0 USEFUL: 0 USELESS: 0
cpu0->LLC AVERAGE MISS LATENCY: 152.7 cycles

DRAM Statistics

Channel 0 RQ ROW_BUFFER_HIT: 451
ROW_BUFFER_MISS: 32535
AVG DBUS CONGESTED CYCLE: 14.58
Channel 0 WQ ROW_BUFFER_HIT: 1986
ROW_BUFFER_MISS: 23739
FULL: 0
Channel 0 REFRESHES ISSUED: 3421

```

| پارامترهای خواسته شده:

:IPC (Instruction Per Cycle) - 1

```
CPU 0 cumulative IPC: 0.9744 instructions: 40000002 cycles: 41050392
```

یعنی در هر سیکل به طور میانگین 0.9744 دستور اجرا شده است.

:LLC Hit Rate و Miss Rate - 2

Access:	140359
Hit:	104426
Miss:	35933

$$* \text{ Hit Rate: } \frac{104426}{140359} \approx 74.4\%$$

$$* \text{ Miss Rate: } \frac{35933}{140359} \approx 25.6\%$$

:Average Miss Latency - 3

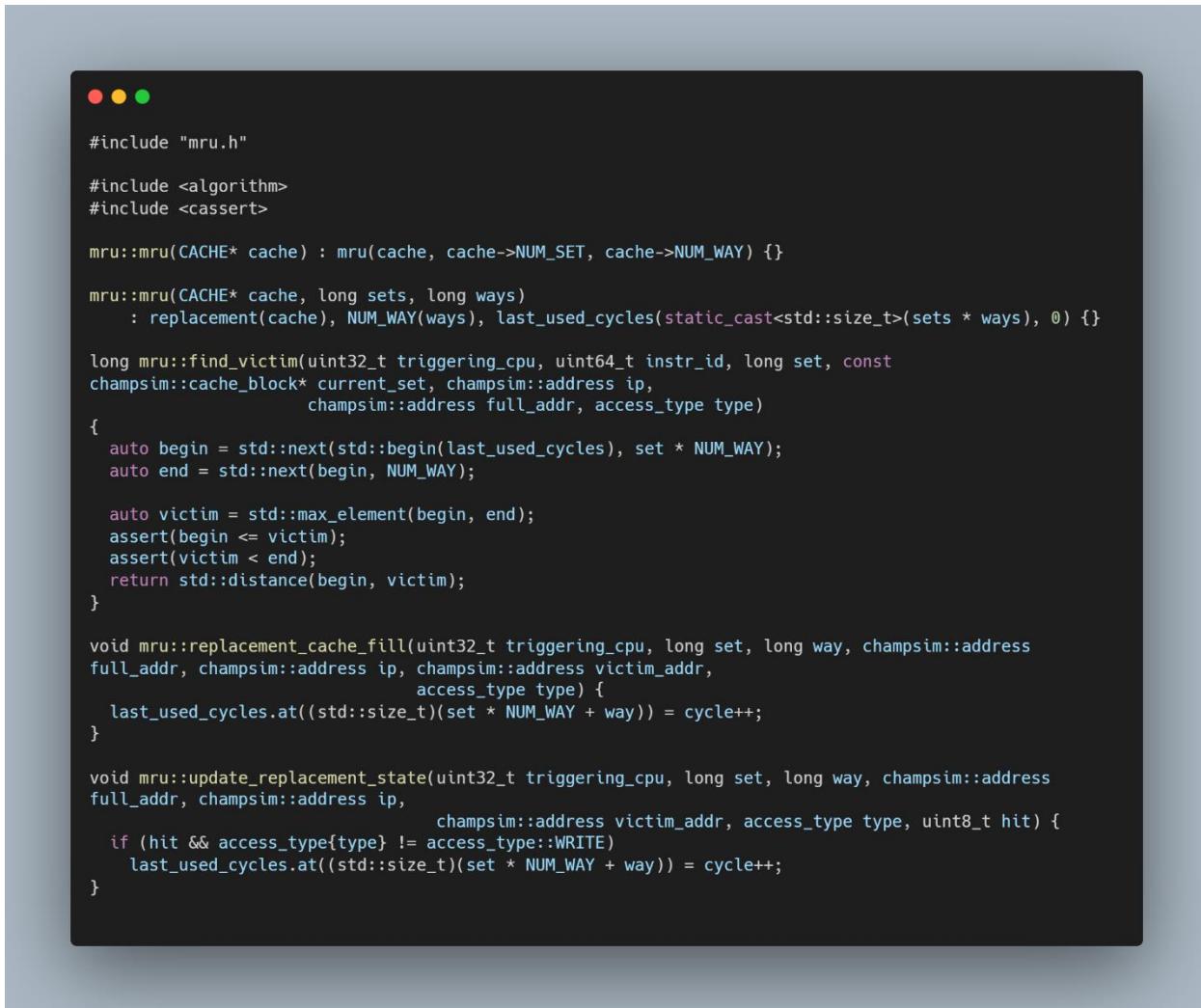
برابر با: 152.7 cycles

- باتوجه به نتایج می‌توان گفت که FIFO نمی‌تواند داده‌ها را به خوبی مدیریت کند و مداوماً به حافظه اصلی یا DRAM دسترسی پیدا می‌کند.

د) پیاده‌سازی الگوریتم MRU:

کارکردی دقیقاً خلاف LRU دارد. یعنی جای حذف قدیمی‌ترین بلوک استفاده شده، جدیدترین بلوک مورد استفاده را حذف می‌کند.

| فایل :mru.cc



```
#include "mru.h"

#include <algorithm>
#include <cassert>

mru::mru(CACHE* cache) : mru(cache, cache->NUM_SET, cache->NUM_WAY) {}

mru::mru(CACHE* cache, long sets, long ways)
    : replacement(cache), NUM_WAY(ways), last_used_cycles(static_cast<std::size_t>(sets * ways), 0) {}

long mru::find_victim(uint32_t triggering_cpu, uint64_t instr_id, long set, const
                      champsim::cache_block* current_set, champsim::address ip,
                      champsim::address full_addr, access_type type)
{
    auto begin = std::next(std::begin(last_used_cycles), set * NUM_WAY);
    auto end = std::next(begin, NUM_WAY);

    auto victim = std::max_element(begin, end);
    assert(begin <= victim);
    assert(victim < end);
    return std::distance(begin, victim);
}

void mru::replacement_cache_fill(uint32_t triggering_cpu, long set, long way, champsim::address
                                 full_addr, champsim::address ip, champsim::address victim_addr,
                                 access_type type) {
    last_used_cycles.at((std::size_t)(set * NUM_WAY + way)) = cycle++;
}

void mru::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 (hit && access_type{type} != access_type::WRITE)
        last_used_cycles.at((std::size_t)(set * NUM_WAY + way)) = cycle++;
}
```

| فایل :mru.h

```

#ifndef REPLACEMENT_MRU_H
#define REPLACEMENT_MRU_H

#include <vector>
#include "cache.h"
#include "modules.h"

class mru : public champsim::modules::replacement
{
    long NUM_WAY;
    std::vector<uint64_t> last_used_cycles;
    uint64_t cycle = 0;

public:
    explicit mru(CACHE* cache);
    mru(CACHE* cache, long sets, long ways);

    long find_victim(uint32_t triggering_cpu, uint64_t instr_id, long set, const champsim::cache_block*
current_set, champsim::address ip,
                    champsim::address full_addr, access_type type);
    void replacement_cache_fill(uint32_t triggering_cpu, long set, long way, champsim::address full_addr,
champsim::address ip, champsim::address victim_addr,
                    access_type type);
    void 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);
};

#endif

```

نتایج به دست آمده در ترمینال بعد از اتمام شبیه سازی در سطح LLC (بقیه سطوح مشابه LRU)

```

rein@DESKTOP-C4TMSM1:~/ChampSim$ ./bin/champsim --warmup-instructions 60000000 --simulation-instructions 40000000 ~/traces/astar_163B.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: 10000002 cycles: 3327685 heartbeat IPC: 3.005 cumulative IPC: 3.005 (Simulation time: 00 hr 01 min 09 sec)
Heartbeat CPU 0 instructions: 20000004 cycles: 6068533 heartbeat IPC: 3.649 cumulative IPC: 3.296 (Simulation time: 00 hr 02 min 11 sec)
Heartbeat CPU 0 instructions: 30000004 cycles: 8630466 heartbeat IPC: 3.903 cumulative IPC: 3.476 (Simulation time: 00 hr 03 min 09 sec)
Heartbeat CPU 0 instructions: 40000006 cycles: 11150109 heartbeat IPC: 3.969 cumulative IPC: 3.587 (Simulation time: 00 hr 04 min 07 sec)
Heartbeat CPU 0 instructions: 50000010 cycles: 13650604 heartbeat IPC: 3.999 cumulative IPC: 3.663 (Simulation time: 00 hr 05 min 04 sec)
Warmup finished CPU 0 instructions: 60000000 cycles: 16154783 cumulative IPC: 3.714 (Simulation time: 00 hr 06 min 01 sec)
Warmup complete CPU 0 instructions: 60000000 cycles: 16154783 cumulative IPC: 3.714 (Simulation time: 00 hr 06 min 01 sec)
Heartbeat CPU 0 instructions: 60000012 cycles: 16154786 heartbeat IPC: 3.993 cumulative IPC: 4 (Simulation time: 00 hr 06 min 01 sec)
Heartbeat CPU 0 instructions: 70000016 cycles: 26891343 heartbeat IPC: 0.9314 cumulative IPC: 0.9314 (Simulation time: 00 hr 07 min 54 sec)
Heartbeat CPU 0 instructions: 80000018 cycles: 38282178 heartbeat IPC: 0.8779 cumulative IPC: 0.9039 (Simulation time: 00 hr 09 min 52 sec)
Heartbeat CPU 0 instructions: 90000018 cycles: 49455574 heartbeat IPC: 0.895 cumulative IPC: 0.9009 (Simulation time: 00 hr 11 min 48 sec)
Simulation finished CPU 0 instructions: 40000002 cycles: 44576057 cumulative IPC: 0.8973 (Simulation time: 00 hr 13 min 43 sec)
Simulation complete CPU 0 instructions: 40000002 cycles: 44576057 cumulative IPC: 0.8973 (Simulation time: 00 hr 13 min 43 sec)

ChampSim completed all CPUs

==== Simulation ====
CPU 0 runs /home/rein/traces/astar_163B.trace.xz

Region of Interest Statistics

CPU 0 cumulative IPC: 0.8973 instructions: 40000002 cycles: 44576057
CPU 0 Branch Prediction Accuracy: 77.23% MPKI: 33.35 Average ROB Occupancy at Mispredict: 14.83
Branch type MPKI
BRANCH_DIRECT_JUMP: 0
BRANCH_INDIRECT: 0
BRANCH_CONDITIONAL: 33.35
BRANCH_DIRECT_CALL: 0
BRANCH_INDIRECT_CALL: 0
BRANCH_RETURN: 0

```

```

cpu0->LLC TOTAL ACCESS: 140360 HIT: 20572 MISS: 119788 MSHR_MERGE: 0
cpu0->LLC LOAD ACCESS: 47305 HIT: 7605 MISS: 39700 MSHR_MERGE: 0
cpu0->LLC RFO ACCESS: 28794 HIT: 6560 MISS: 22234 MSHR_MERGE: 0
cpu0->LLC PREFETCH ACCESS: 0 HIT: 0 MISS: 0 MSHR_MERGE: 0
cpu0->LLC WRITE ACCESS: 60115 HIT: 5994 MISS: 54121 MSHR_MERGE: 0
cpu0->LLC TRANSLATION ACCESS: 4146 HIT: 413 MISS: 3733 MSHR_MERGE: 0
cpu0->LLC PREFETCH REQUESTED: 0 ISSUED: 0 USEFUL: 0 USELESS: 0
cpu0->LLC AVERAGE MISS LATENCY: 99 cycles

DRAM Statistics

Channel 0 RQ ROW_BUFFER_HIT: 1302
ROW_BUFFER_MISS: 64365
AVG DBUS CONGESTED CYCLE: 19.15
Channel 0 WQ ROW_BUFFER_HIT: 8090
ROW_BUFFER_MISS: 50293
FULL: 0
Channel 0 REFRESHES ISSUED: 3714
rein@DESKTOP-C4TMSM1:~/ChampSim$
```

| پارامترهای خواسته شده:

:IPC (Instruction Per Cycle) - 1

```
CPU 0 cumulative IPC: 0.8973 instructions: 40000002 cycles: 44576057
```

عنی در هر سیکل به طور میانگین 0.8973 دستور اجرا شده است.

:LLC Hit Rate و Miss Rate - 2

Access:	140360
Hit:	20572
Miss:	119788

$$* \text{ Hit Rate: } \frac{20572}{140360} \approx 14.7\%$$

$$* \text{ Miss Rate: } \frac{119788}{140360} \approx 85.3\%$$

الگوریتم MRU به طور کلی در کش‌های بزرگتر مثل همین LLC عملکرد خوبی ندارد. با توجه به اینکه MRU براساس داده‌های جدیدتر عمل می‌کند، باعث می‌شود که این داده‌های جدید به سرعت از کش حذف شوند در حالی که داده‌های قدیمی‌تر که ممکن است کمتر مورد استفاده قرار بگیرند در کش باقی بمانند.

:Average Miss Latency - 3

برابر با: 99 cycles

- این مقدار نسبت به FIFO و LFU بهینه‌تر است.

مقایسه نهایی Replacement Policy ها

	LRU	LFU	FIFO	MRU
IPC	0.9749	0.9065	0.9744	0.8973
Miss Rate	23.6%	74.7%	25.6%	85.3%
Miss Latency	166.5	105.3	152.7	99

Miss Rate -

الگوریتم MRU با Miss Rate بسیار بالا (حدود ۸۵,۳ درصد) نسبت به بقیه سیاست‌ها عملکرد ضعیفی در LLC دارد. یعنی توانایی نگهداری داده‌های لازم را در کش ندارد و نیاز بیشتری به دسترسی به حافظه اصلی پیدا می‌شود. بعد از MRU، بیشترین Miss Rate متعلق به الگوریتم LFU (با حدود ۷۴,۷ درصد) می‌باشد.

سیاست‌های LRU و FIFO برای کش‌هایی مثل LLC بهتر هستند ولی همچنان بهینه نیستند.

IPC -

IPC و FIFO دارای LRU بیشتری هستند که یعنی این سیاست‌ها از نظر سرعت عملکرد بهتری دارند؛ چون تعداد دستور بیشتری را در یک سیکل پردازش می‌کنند. از طرفی IPC در MRU و LFU کمتر است که یعنی نسبتاً کندر هستند و کارایی کمتری در پردازش دستورات دارند.

=> در حالت کلی، در این شرایط و در کش LLC، سیاست‌های FIFO و LRU عملکرد بهتری داشتند که یعنی برای کش‌های بزرگتر مناسب‌تر هستند. (IPC و LFU باعث کاهش کارایی کلی آن‌ها می‌شود).

[بخش دوم پروژه، قسمت امتیازی]

به طور کلی هدف مقاله داده شده ارائه یک replacement policy بهینه است که در شرایط واقعی و در ابعاد بزرگتر موثر باشد. یعنی:

سیاست (Estimated Time of Arrival) ETA

منطق اصلی این سیاست این است که در هر لحظه تخمین زده شود که کدام بلوک کش دیرتر از بقیه بلوکها دوباره مورد استفاده قرار می‌گیرد که در هنگام نیاز آن را حذف کنیم. (زمان تقریبی استفاده مجدد از یک cache (line

فرمول تعریف شده برای ETA به صورت زیر است:

$$\text{ETA}(\text{line}) = \text{current_time} + \text{predicted_reuse_distance}(\text{line})$$

که در آن current time شمارندهای از زمان کنونی اجرای برنامه است و فاصله predicted_reuse_distance تقریبی بین استفاده کنونی از یک لاین و استفاده بعدی از آن است. هرچقدر مقدار محاسبه شده برای ETA بیشتر باشد، به این معنی است که بلوک مدنظر دیرتر استفاده می‌شود و برای حذف کردن مناسب‌تر است.

برای پیاده‌سازی این سیاست و استفاده از آن برای شبیه‌سازی، برای هر بلوک از کش باید یک مقدار ETA نگهداری کنیم. زمانی که کش پر شد و نیاز به جایگزینی داشتیم باید بین همه بلوک‌های موجود در کش، بلوکی را پیدا کنیم که بیشترین مقدار ETA را دارد و حذف کنیم؛ سپس بلوک جدید را وارد کش کرده و مقدار ETA آن را نیز محاسبه و نگهداری کنیم.

* عبارت reuse distance به معنی فاصله بین دو دفعه‌ای است که به یک آدرس خاص (یک بلوک یا لاین خاص) دسترسی صورت گرفته.

```
● ● ●

#include "eta.h"
#include <algorithm>
#include <cassert>

eta::eta(CACHE* cache) : eta(cache, cache->NUM_SET, cache->NUM_WAY) {}

eta::eta(CACHE* cache, long sets, long ways)
    : replacement(cache), NUM_WAY(ways), eta_value(static_cast<std::size_t>(sets * ways), 0) {}

long eta::find_victim(uint32_t triggering_cpu, uint64_t instr_id, long set, const
champsim::cache_block* current_set, champsim::address ip,
                      champsim::address full_addr, access_type type) {
    auto begin = std::next(eta_value.begin(), set * NUM_WAY);
    auto end = std::next(begin, NUM_WAY);
    auto victim = std::max_element(begin, end);

    assert(begin <= victim && victim < end);
    return std::distance(begin, victim);
}

void eta::replacement_cache_fill(uint32_t triggering_cpu, long set, long way, champsim::address
full_addr, champsim::address ip, champsim::address victim_addr,
                                 access_type type) {
    uint64_t full_addr_val = full_addr.to<uint64_t>();
    uint64_t predicted_reuse = (full_addr_val % 1000) + 100;
    eta_value.at(set * NUM_WAY + way) = current_time + predicted_reuse;
    current_time++;
}

void eta::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 (hit) {
        uint64_t full_addr_val = full_addr.to<uint64_t>();
        eta_value.at(set * NUM_WAY + way) = current_time + ((full_addr_val % 1000) + 100);
    }
    current_time++;
}
```

```

#ifndef REPLACEMENT_ETA_H
#define REPLACEMENT_ETA_H

#include <vector>
#include "cache.h"
#include "modules.h"

class eta : public champsim::modules::replacement
{
    long NUM_SET;
    long NUM_WAY;
    std::vector<uint64_t> eta_value;
    uint64_t current_time = 0;

public:
    explicit eta(CACHE* cache);
    eta(CACHE* cache, long sets, long ways);

    long find_victim(uint32_t triggering_cpu, uint64_t instr_id, long set, const champsim::cache_block* current_set, champsim::address ip,
                     champsim::address full_addr, access_type type);
    void replacement_cache_fill(uint32_t triggering_cpu, long set, long way, champsim::address full_addr,
                                champsim::address ip, champsim::address victim_addr,
                                access_type type);
    void 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);
};

#endif

```

```

rein@DESKTOP-C4TMSM1:~/ChampSim$ ./bin/champsim --warmup-instructions 60000000 --simulation-instructions 40000000 ~/traces/astar_163B.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: 3327675 heartbeat IPC: 3.005 cumulative IPC: 3.005 (Simulation time: 00 hr 00 min 50 sec)
Heartbeat CPU 0 instructions: 20000004 cycles: 6068563 heartbeat IPC: 3.648 cumulative IPC: 3.296 (Simulation time: 00 hr 01 min 39 sec)
Heartbeat CPU 0 instructions: 30000000 cycles: 8630503 heartbeat IPC: 3.903 cumulative IPC: 3.476 (Simulation time: 00 hr 02 min 26 sec)
Heartbeat CPU 0 instructions: 40000010 cycles: 11150125 heartbeat IPC: 3.969 cumulative IPC: 3.587 (Simulation time: 00 hr 03 min 15 sec)
Heartbeat CPU 0 instructions: 50000010 cycles: 13650614 heartbeat IPC: 3.999 cumulative IPC: 3.663 (Simulation time: 00 hr 04 min 02 sec)
Warmup complete CPU 0 instructions: 60000000 cycles: 16154793 cumulative IPC: 3.714 (Simulation time: 00 hr 04 min 44 sec)
Warmup finished CPU 0 instructions: 60000000 cycles: 16154793 cumulative IPC: 3.714 (Simulation time: 00 hr 04 min 44 sec)
Heartbeat CPU 0 instructions: 60000012 cycles: 16154796 heartbeat IPC: 3.993 cumulative IPC: 4 (Simulation time: 00 hr 04 min 44 sec)
Heartbeat CPU 0 instructions: 70000016 cycles: 26909383 heartbeat IPC: 0.9298 cumulative IPC: 0.9298 (Simulation time: 00 hr 06 min 13 sec)
Heartbeat CPU 0 instructions: 80000018 cycles: 38320443 heartbeat IPC: 0.8763 cumulative IPC: 0.9023 (Simulation time: 00 hr 07 min 38 sec)
Heartbeat CPU 0 instructions: 90000018 cycles: 49525868 heartbeat IPC: 0.8924 cumulative IPC: 0.899 (Simulation time: 00 hr 09 min 05 sec)
Simulation finished CPU 0 instructions: 40000002 cycles: 44644378 cumulative IPC: 0.896 (Simulation time: 00 hr 10 min 31 sec)
Simulation complete CPU 0 instructions: 40000002 cycles: 44644378 cumulative IPC: 0.896 (Simulation time: 00 hr 10 min 31 sec)

ChampSim completed all CPUs

*** Simulation ===
CPU 0 runs /home/rein/traces/astar_163B.trace.xz

Region of Interest Statistics

CPU 0 cumulative IPC: 0.896 instructions: 40000002 cycles: 44644378
CPU 0 Branch Prediction Accuracy: 77.23% MPKI: 33.35 Average ROB Occupancy at Mispredict: 14.8Branch type MPKI
BRANCH_DIRECT_JUMP: 0
BRANCH_INDIRECT: 0
BRANCH_CONDITIONAL: 33.35
BRANCH_DIRECT_CALL: 0
BRANCH_INDIRECT_CALL: 0
BRANCH_RETURN: 0

```

ناتیج بهدست آمده در ترمینال بعد از اتمام شبیه‌سازی در سطح LLC (بقیه مشابه سطوح LRU)

```

cpu0->cpu0_L2C TOTAL ACCESS: 230491 HIT: 150245 MISS: 80246 MSHR_MERGE: 0
cpu0->cpu0_L2C LOAD ACCESS: 111727 HIT: 64422 MISS: 47305 MSHR_MERGE: 0
cpu0->cpu0_L2C RFO ACCESS: 30622 HIT: 1828 MISS: 28794 MSHR_MERGE: 0
cpu0->cpu0_L2C PREFETCH ACCESS: 0 HIT: 0 MISS: 0 MSHR_MERGE: 0
cpu0->cpu0_L2C WRITE ACCESS: 83905 HIT: 83904 MISS: 1 MSHR_MERGE: 0
cpu0->cpu0_L2C TRANSLATION ACCESS: 4237 HIT: 91 MISS: 4146 MSHR_MERGE: 0
cpu0->cpu0_L2C PREFETCH REQUESTED: 0 ISSUED: 0 USEFUL: 0 USELESS: 0
cpu0->cpu0_L2C AVERAGE MISS LATENCY: 160.1 cycles
cpu0->cpu0_L1I TOTAL ACCESS: 3894 HIT: 3894 MISS: 0 MSHR_MERGE: 0
cpu0->cpu0_L1I LOAD ACCESS: 3894 HIT: 3894 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: 11299458 HIT: 11038262 MISS: 261196 MSHR_MERGE: 114610
cpu0->cpu0_L1D LOAD ACCESS: 7164870 HIT: 7035918 MISS: 128952 MSHR_MERGE: 17225
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: 4129632 HIT: 4001631 MISS: 128001 MSHR_MERGE: 97379
cpu0->cpu0_L1D TRANSLATION ACCESS: 4956 HIT: 713 MISS: 4243 MSHR_MERGE: 6
cpu0->cpu0_L1D PREFETCH REQUESTED: 0 ISSUED: 0 USEFUL: 0 USELESS: 0
cpu0->cpu0_L1D AVERAGE MISS LATENCY: 96.21 cycles
cpu0->cpu0_ITLB TOTAL ACCESS: 3425 HIT: 3425 MISS: 0 MSHR_MERGE: 0
cpu0->cpu0_ITLB LOAD ACCESS: 3425 HIT: 3425 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: 11001257 HIT: 10865413 MISS: 135844 MSHR_MERGE: 9125
cpu0->cpu0_DTLB LOAD ACCESS: 11001257 HIT: 10865413 MISS: 135844 MSHR_MERGE: 9125
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: 11.26 cycles
cpu0->LLC TOTAL ACCESS: 140360 HIT: 21766 MISS: 118594 MSHR_MERGE: 0
cpu0->LLC LOAD ACCESS: 47305 HIT: 7271 MISS: 40034 MSHR_MERGE: 0
cpu0->LLC RFO ACCESS: 28794 HIT: 5629 MISS: 23165 MSHR_MERGE: 0
cpu0->LLC PREFETCH ACCESS: 0 HIT: 0 MISS: 0 MSHR_MERGE: 0
cpu0->LLC WRITE ACCESS: 60115 HIT: 8315 MISS: 51800 MSHR_MERGE: 0
cpu0->LLC TRANSLATION ACCESS: 4146 HIT: 551 MISS: 3595 MSHR_MERGE: 0
cpu0->LLC PREFETCH REQUESTED: 0 ISSUED: 0 USEFUL: 0 USELESS: 0

DRAM Statistics

Channel 0 RQ ROW_BUFFER_HIT: 1173
    ROW_BUFFER_MISS: 65621
    AVG DBUS CONGESTED CYCLE: 18.97
Channel 0 WQ ROW_BUFFER_HIT: 7810
    ROW_BUFFER_MISS: 50198
    FULL: 0
Channel 0 REFRESHES ISSUED: 3720
rein@DESKTOP-C4TMSM1:~/ChampSim$
```

| پارامترهای خواسته شده:

:IPC (Instruction Per Cycle) - 1

CPU 0 cumulative IPC: 0.896 instructions: 40000002 cycles: 44644378

یعنی در هر سیکل به طور میانگین ۰.۸۹۶ دستور اجرا شده است.

Hit Rate و Miss Rate - 2 در لایه‌های مختلف کش:

یا کش لایه آخر): LLC - Last Level Cache)

Access:	140360
Hit:	21766
Miss:	118594

$$* \text{ Hit Rate: } \frac{21766}{140360} \approx 15.5\%$$

$$* \text{ Miss Rate: } \frac{118594}{140360} \approx 84.5\%$$

:L1D -

Access: 11299458
Hit: 11038262
Miss: 261196

$$* \text{ Hit Rate: } \frac{11038262}{11299458} \approx 97.7\%$$

$$* \text{ Miss Rate: } \frac{261196}{11299458} \approx 2.3\%$$

عملکرد خوبی برای L1D است. معمولا hit rate در این سطح بالاست چون دسترسی به لایه اول یا سریع‌ترین نوع دسترسی است.

:L2C -

Access: 230491
Hit: 150245
Miss: 80246

$$* \text{ Hit Rate: } \frac{150245}{230491} \approx 65.2\%$$

$$* \text{ Miss Rate: } \frac{80246}{230491} \approx 34.8\%$$

Average Miss Latency - 3

برابر با: 101.6 cycles

- هرچقدر miss rate بیشتر باشد، داده‌ها باید از سطوح پایین‌تر حافظه بارگذاری شوند که زمان دسترسی را افزایش و IPC را کاهش می‌دهد. به طور مثال، در LLC به خاطر miss rate بالا (یعنی در حدود ۸۴.۵ درصد) داده‌ها باید از حافظه اصلی بارگذاری شوند که می‌تواند باعث کاهش IPC شود. از طرفی، miss rate پایین در L1D، یعنی بیشتر داده‌ها از این لایه خوانده می‌شوند که IPC را افزایش و منجر به بیشتر شدن کارایی سیستم می‌شود.

| مقایسه با سیاست‌های دیگر:

- ETA: با استفاده از پیش‌بینی زمان استفاده مجدد داده‌ها، باعث کم شدن miss rate و عملکرد بهتر در کش‌هایی که miss rate بالا دارند می‌شود. به همین دلیل برای داده‌هایی که دسترسی‌های بیشتری در

کش دارند مناسب است. از طرفی، به خاطر داشتن هزینه محاسباتی بیشتر می‌تواند کارایی و IPC را تحت تاثیر قرار داده و آن‌ها را کاهش دهد. همچنین اگر miss rate در مثلا L1D کم باشد، ممکن است خوب عمل نکند.

- IPC: معمولاً IPC بهتری نسبت به بقیه سیاست‌ها دارد و پیاده‌سازی آن نیز راحت است. زمانی که در کش دسترسی‌ها از الگوی خاصی پیروی نکنند، ممکن است که بهینه عمل نکند و miss rate آن افزایش پیدا کند.

- LFU: درباره داده‌هایی که استفاده زیادی دارند خوب عمل می‌کند؛ اما عملکردش در مواجهه با داده‌هایی که تغییرات آنها زیاد است ضعیف می‌باشد. همچنین به خاطر نگهداری از تعداد استفاده از بلوک‌های کش، به حافظه بیشتری نیاز دارد و پیچیدگی محاسبه آن هم زیاد می‌شود.

- MRU: پیاده‌سازی راحتی دارد و الگوریتمش هم سریع عمل می‌کند. اگر داده‌ای زیاد مورد استفاده قرار نگیر خوب عمل می‌کند. از طرفی miss rate آن بسیار بالاست که باعث کم شدن IPC می‌شود.

- FIFO: اگر از داده‌های قدیمی استفاده نشود عملکرد مناسبی دارد. توجه نکردن به زمان دسترسی به داده‌ها یا تعداد استفاده از آن‌ها، بهینه بودن عملکردش را تحت تاثیر قرار می‌دهد.

| پیچیدگی سخت‌افزاری و هزینه پیاده‌سازی سیاست ETA:

- برای ذخیره‌سازی ETA‌ها به خاطر نیاز به نگهداری از آدرس‌های داده‌ها و همچنین شمارنده‌هایی برای محاسبه فاصله بین دسترسی‌ها، نیاز به حافظه اضافی وجود دارد که باعث افزایش هزینه‌های سخت‌افزاری می‌شود.

- محاسبه ETA‌های داده‌ها با هر بار دسترسی به حافظه اصلی، نیازمند پردازش اضافی است که باعث زیاد شدن زمان پاسخگویی می‌شود و هزینه‌های محاسباتی بیشتری را ایجاد می‌کند. (چون محاسبه ETA به الگوریتم پیش‌بینی نیاز دارد که باید به صورت مداوم آپدیت شود).

- در مقایسه با بقیه سیاست‌های معرفی شده که پیچیدگی کمتری دارند، این سیاست می‌تواند در بعضی از شرایط خاص عملکرد بهتری داشته باشد ولی هزینه‌ها و پیچیدگی‌های بیشتری هم دارد.