دانشگاه صنعتی شریف
دانشکده مهندسی کامپیوتر

کارشناسی
مهندسی کامپیوتر

پروژه ی معماری کامپیوتر

نگارش

مهدی علی نژاد، محمدامین عباسفر، سعید فراتی، امیرحسین صوری
گروه ۷

استاد راهنما

دکتر اسدی

بهار ۱۴۰۳

پیاده سازی Cache replacement initating Belady's OPT policy در Gem۵

در این پروژه قصد داریم یک سیاست جایگزینی حافظه نهان را به Gem5 اضافه کنیم و با اجرای یک برنامه محک عملکرد آن را با سیاست‌های موجود مقایسه کنیم.

سیاست جایگزینی معرفی شده در مقاله

به طور کلی برای هر خط از حافظه نهان، زمان تقریبی رسیدن به آن خط یا ETA برابر جمع زمان کنونی و پارامتری به اسم Predicted reuse distance است. در هر اضافه کردن یک خط به حافظه نهان، خط با بالاترین ETA از حافظه نهان، evict می‌شود.

وظایف:

- یادگیری طرز کار الگوریتم گفته شده و مطالعه ی مقاله ی مربوطه
- پیاده سازی الگوریتم و قرار دادن آن در دل Gem۵
- پیاده سازی یک سری برنامه ی محک و تست کردن سیاست گفته شده
- مقایسه ی عملکرد آن با سیاست های معروف مانند LFU و LRU

توضیحاتی در خصوص نحوه پیاده سازی

در این قسمت توضیحاتی در خصوص نحوه پیاده سازی توابع مختلف در این replacement policy میدهیم:

Invalidate

```
void
Belady::invalidate(const std::shared_ptr<ReplacementData>& replacement_data
    ) {
    if (!replacement_data) {
        std::cerr << "invalidate: replacement_data is null!" << std::endl;
        return;
    }

    auto rd = std::static_pointer_cast<BeladyReplData>(replacement_data);
    if (!rd) {
        std::cerr << "invalidate: BeladyReplData is null!" << std::endl;
        return;
    }

    rd->ETR = 1 << 31;
}
```

زمانی که این تابع برای بلوکی فراخوانده می شود، ETR مربوط به این بلوک را برابر بیشنینه زمان ممکن میگذاریم تا به عنوان بدترین بلوک شناخته شده و در اولین فرصت جایگزین شود.

Touch

```
void
Belady::touch(const std::shared_ptr<ReplacementData>& replacement_data)
    const {
    if (!replacement_data) {
        std::cerr << "touch: replacement_data is null!" << std::endl;
        return;
```

```
    }

    auto rd = std::static_pointer_cast<BeladyReplData>(replacement_data);
    if (!rd || !rd->pkt || !rd->pkt->req) {
        std::cerr << "touch: BeladyReplData or its packet/request is null!"
    << std::endl;
        return;
    }
    Addr pc_addr = 0;
    if (rd->pkt->req->hasPC())
        pc_addr = rd->pkt->req->getPC();
    Addr addr = rd->pkt->getBlockAddr(7);
    long reused = curTick() - rd->last_touched;
    access_train_data(pc_addr, addr, reused, rd->ETR);
    rd->ETR = Belady::getERD(pc_addr, addr);
}
```

با هر دسترسی به یک بلوک، این تابع فراخوانی می شود. ابتدا مقدار pc و آدرس بلوک فعلی را محاسبه می نماییم. همچنین برای بروز رسانی RDP نیاز به محاسبه فاصله استفاده مجدد هستیم که این کار را با تفریق زمان فعلی از زمان قبلی ای که برای این بلوک ثبت شده انجام می دهیم. می دانیم در هردسترسی بلوک دوکار باید انجام دهیم. ۱- بروز رسانی sampled cache و آموزش دادن RDP و ۲- reset کردن ETR بلوک در حافظه نهان اصلی مورد اول را با تابع access_train_data انجام می دهیم (که بعدا به توضیح آن می پردازیم) و مورد دوم در همین تابع انجام میگیرد و مقدار درون RDP برای این بلوک را مجددا درون این شمارنده میریزد

Reset

```
void
Belady::reset(const std::shared_ptr<ReplacementData>& replacement_data)
    const {
    auto rd = std::static_pointer_cast<BeladyReplData>(replacement_data);
    rd->ETR = std::numeric_limits<long long>::max();
    rd->last_touched = curTick();
}
```

در این بلوک که برای بلوک هایی است که به تازگی initialize می شوند، مقدار last_touched را برابر زمان فعلی میگذاریم تا بعدا بتوانیم reuse distance را برای آنها محاسبه کنیم

getVictim

```
ReplaceableEntry* Belady::getVictim(const ReplacementCandidates& candidates) const {
    assert(candidates.size() > 0);

    ReplaceableEntry* victim = candidates[0];
    auto victim_rd = std::static_pointer_cast<BeladyReplData>(victim->replacementData);
    if (!victim_rd) {
        std::cerr << "getVictim: initial victim replacementData is null!" << std::endl;
        return victim;
    }

    long max_reused = abs((victim_rd->last_touched + victim_rd->ETR) - curTick());

    for (const auto& candidate : candidates) {
        auto rd = std::static_pointer_cast<BeladyReplData>(candidate->replacementData);
        if (!rd) {
            std::cerr << "getVictim: candidate replacementData is null!" << std::endl;
            continue;
        }

        long reused = abs((rd->last_touched + rd->ETR) - curTick());
        if (reused > max_reused) {
            max_reused = reused;
            victim = candidate;
        }
```

```
    }

    return victim;
}
```

این تابع زمانی صدا می شود که میخواهیم بلوکی را حافظه نهان بیرون بیاندازیم و دنبال بی ارزش ترین بلوک میگردیم. در این replacement policy معیار بی ارزش بودن، دور بودن reuse distance از زمان فعلی است. برای چک کردن این مورد، تمام بلوک ها را بررسی میکنیم و بلوکی که بیشترین ETR را از لحاظ اندازه (چه مثبت و چه منفی) دارد را برمیگردانیم

BeladyReplData

```
struct BeladyReplData : ReplacementData {
    long long ETR, last_touched;
    bool first_time = true;
    BeladyReplData() {}
};
```

در این داده ساختارکه اطلاعاتی است که یک بلوک نگهداری میکند، داده هایی از جمله ETR و PC ای که آخرین دسترسی را به این بلوک داشته را ذخیره میکنیم تا بعدا بتوانیم با توجه به این اطلاعات RDP را آموزش دهیم

update_RDP

```
long update_RDP(long dist, long prev_value, float lr) const {
    long diff_sign = dist < 0 ? -1 : 1;
    double diff = diff_sign * dist;
    double update_value = diff_sign * (diff * lr < 10 ? 10 : diff * lr);
    long new_value = update_value + prev_value;
    return new_value;
}
```

در این تابع از الگوریتم temporal difference learning برای آموزش مقادیر درون RDP استفاده میکنیم. به این صورت که هر دفعه به جای اینکه مقدار RDP را برابر reuse distance

جدید بدست آمده قرار دهیم، به اندازه اختلاف این دو ضرب تر learning rate ، مقدار قدیمی را بروزرسانی مینماییم

access_train_data

```
void access_train_data(unsigned int new_pc, Addr addr,long reused, long
    prev_value) const {
    long new_val = update_RDP(reused, prev_value, get_learning_rate(new_pc,
    addr));
    if (hist_rdp.find({addr, new_pc}) == hist_rdp.end())
        hist_rdp[{addr, new_pc}] = 0;
    else
        hist_rdp[{addr, new_pc}] = new_val;
}
```

در این تابع کار های مربوط به فرایند training را انجام مدهیم. در ابتدا تابع update_RDP را فرا میخوانیم تا کار های مربوط به آن قسمت انجام شود. سپس در ادامه مقادیر ذخیره شده در sampled cache را بروزرسانی میکنیم

get_learning_rate

```
float get_learning_rate(unsigned int new_pc, Addr addr) const {
    if (count_updates.find({addr, new_pc}) == count_updates.end())
        count_updates[{addr, new_pc}] = 1;

    int access = count_updates[{addr, new_pc}];
    float cur_rate = (float) 2 / (3 + access);
    count_updates[{addr, new_pc}] += 1;
    return cur_rate > 0.1 ? cur_rate : 0.1;
}
```

در این تابع که برای سریع تر شدن فرایند training نوشته شده، learning rate را با توابعی دلخواه، در طول زمان تغییر میدهیم، و مقدار جدید را خروجی میدهیم.

پیاده سازی الگوریتم در دل gem۵

آماده کردن gem۵

برای شروع کار با gem۵ ابتدا نیاز است آن را clone کنیم، این کار را با دستور زیر انجام می دهیم

```
git clone https://gem5.googlesource.com/public/gem5
```

Listing 1: cloning gem5

اضافه کردن تغییرات لازمه

برای این الگوریتم ما نیاز به مقدار PC درخواست دهنده ی cache request هستیم، آنرا به این صورت در دسترس الگوریتم قرار می دهیم
ابتدا به فایل

gem5/src/mem/cache/replacement_policies/replaceable_entry.hh

رفته و کد زیر را به replacementData اضافه می کنیم

```
struct ReplacementData {
    PacketPtr pkt;
};
```

Listing 2: replacementData

سپس به gem۵/src/mem/cache/base.hh رفته و در توابع آن قبل از پاس دادن replacement_data به آن اطلاعاتی که نیاز داریم را هم اضافه می کنیم. پس از این تغییرات، ما می توانیم با داشتن replacement_data به مقدار PC درخواست دهنده ی آن دسترسی داشته باشیم.

اضافه کردن خود الگوریتم

طبق مراحل گفته شده در صورت پروژه عمل می کنیم.

برای اضافه کردن یکی سیاست جدید به Gem5 لازم است که کد آن را تغییر دهید و سپس دوباره build کنید.

ابتدا در آدرس /src/mem/cache/replacement_policies/ دو دستور زیر را اجرا کنید:

```
        cp lru_rp.cc [policy]_rp.cc
        cp lru_rp.hh [policy]_rp.hh
```

و سپس فایل‌های جدید را با توجه به اسم سیاست جایگزینی تغییر دهید.

سپس به فایل ReplacementPolicies.py در آدرس /src/mem/cache/replacement_policies خطوط زیر را اضافه کنید:

```
        class [POLICY]RP(BaseReplacementPolicy):
        type = '[POLICY]]RP'
        cxx_class = 'gem5::replacement_policy::[POLICY]'
        cxx_header = "mem/cache/replacement_policies/[policy]_rp.hh"
```

نهایتا خط زیر را در فایل src/mem/cache/replacement_policies/SConscript اضافه کنید:

```
        Source('[policy]_rp.cc')
```

خود الگوریتم نیز به این صورت است:

```
#include "mem/cache/replacement_policies/Belady_rp.hh"

#include <cassert>
#include <memory>
#include <limits>
#include "base/types.hh"
#include "params/BeladyRP.hh"
#include "sim/cur_tick.hh"

namespace gem5
{

namespace replacement_policy
{

Belady::Belady(const Params &p)
  : Base(p)
{
}
```

```
20
21 void
22 Belady::invalidate(const std::shared_ptr<ReplacementData>& replacement_data
       )
23 {
24     if (!replacement_data) {
25         std::cerr << "invalidate: replacement_data is null!" << std::endl;
26         return;
27     }
28
29     auto rd = std::static_pointer_cast<BeladyReplData>(replacement_data);
30     if (!rd) {
31         std::cerr << "invalidate: BeladyReplData is null!" << std::endl;
32         return;
33     }
34
35     rd->ETR = 1 << 31;
36 }
37
38 void
39 Belady::touch(const std::shared_ptr<ReplacementData>& replacement_data)
       const
40 {
41     if (!replacement_data) {
42         std::cerr << "touch: replacement_data is null!" << std::endl;
43         return;
44     }
45
46     auto rd = std::static_pointer_cast<BeladyReplData>(replacement_data);
47     if (!rd || !rd->pkt || !rd->pkt->req) {
48         std::cerr << "touch: BeladyReplData or its packet/request is null!"
       << std::endl;
49         return;
50     }
51     Addr pc_addr = 0;
52     if (rd->pkt->req->hasPC())
```

```
53          pc_addr = rd->pkt->req->getPC();
54      Addr addr = rd->pkt->getBlockAddr(7);
55      long reused = curTick() - rd->last_touched;
56      access_train_data(pc_addr, addr, reused, rd->ETR);
57      rd->ETR = Belady::getERD(pc_addr, addr);
58  }
59
60  void
61  Belady::reset(const std::shared_ptr<ReplacementData>& replacement_data)
        const
62  {
63      auto rd = std::static_pointer_cast<BeladyReplData>(replacement_data);
64      rd->ETR = std::numeric_limits<long long>::max();
65      rd->last_touched = curTick();
66  }
67
68  ReplaceableEntry*
69  Belady::getVictim(const ReplacementCandidates& candidates) const
70  {
71      assert(candidates.size() > 0);
72
73      ReplaceableEntry* victim = candidates[0];
74      auto victim_rd = std::static_pointer_cast<BeladyReplData>(victim->
        replacementData);
75      if (!victim_rd) {
76          std::cerr << "getVictim: initial victim replacementData is null!"
        << std::endl;
77          return victim;
78      }
79
80      long max_reused = abs((victim_rd->last_touched + victim_rd->ETR) -
        curTick());
81
82      for (const auto& candidate : candidates) {
83          auto rd = std::static_pointer_cast<BeladyReplData>(candidate->
        replacementData);
```

```
84          if (!rd) {
85              std::cerr << "getVictim: candidate replacementData is null!" <<
    std::endl;
86              continue;
87          }
88
89          long reused = abs((rd->last_touched + rd->ETR) - curTick());
90          if (reused > max_reused) {
91              max_reused = reused;
92              victim = candidate;
93          }
94      }
95
96      return victim;
97  }
98
99  std::shared_ptr<ReplacementData>
100 Belady::instantiateEntry()
101 {
102     return std::shared_ptr<ReplacementData>(new BeladyReplData());
103 }
104
105 } // namespace replacement_policy
106 } // namespace gem5
107
108
109 #ifndef __MEM_CACHE_REPLACEMENT_POLICIES_Belady_RP_HH__
110 #define __MEM_CACHE_REPLACEMENT_POLICIES_Belady_RP_HH__
111
112 #include "mem/cache/replacement_policies/base.hh"
113
114 namespace gem5
115 {
116
117 struct Params;
118
```

```
119 namespace replacement_policy
120 {
121
122 class Belady : public Base
123 {
124   protected:
125     struct BeladyReplData : ReplacementData
126     {
127       long long ETR, last_touched;
128     bool first_time = true;
129       BeladyReplData() {}
130     };
131
132   public:
133     typedef Params Params;
134     Belady(const Params &p);
135     ~Belady() = default;
136
137     void invalidate(const std::shared_ptr<ReplacementData>&
    replacement_data)
138
    override;
139
140     void touch(const std::shared_ptr<ReplacementData>& replacement_data)
    const
141
    override;
142
143     void reset(const std::shared_ptr<ReplacementData>& replacement_data)
    const
144
    override;
145
146     ReplaceableEntry* getVictim(const ReplacementCandidates& candidates)
    const
```

```
147
    override;
148
149     std::shared_ptr<ReplacementData> instantiateEntry() override;
150
151   mutable std::map<std::pair<int, int>, int> hist_rdp;
152   mutable std::map<std::pair<int, int>, int> count_updates;
153
154     long update_RDP(long dist, long prev_value, float lr) const
155     {
156
157         long diff_sign = dist < 0 ? -1 : 1;
158         double diff = diff_sign * dist;
159         double update_value = diff_sign * (diff * lr < 10 ? 10 : diff * lr)
    ;
160         long new_value = update_value + prev_value;
161         return new_value;
162     }
163
164     void access_train_data(unsigned int new_pc, Addr addr,long reused, long
     prev_value) const
165     {
166     long new_val = update_RDP(reused, prev_value, get_learning_rate(new_pc,
     addr));
167       if (hist_rdp.find({addr, new_pc}) == hist_rdp.end())
168       hist_rdp[{addr, new_pc}] = 0;
169     else
170         hist_rdp[{addr, new_pc}] = new_val;
171     }
172
173   float get_learning_rate(unsigned int new_pc, Addr addr) const
174   {
175     if (count_updates.find({addr, new_pc}) == count_updates.end())
176       count_updates[{addr, new_pc}] = 1;
177
178     int access = count_updates[{addr, new_pc}];
```

```
    float cur_rate = (float) 5 / (8 + access);
    count_updates[{addr, new_pc}] += 1;
    return cur_rate > 0.05 ? cur_rate : 0.05;
  }

    long getERD(unsigned int new_pc, Addr addr) const
  {
      if (hist_rdp.find({addr, new_pc}) == hist_rdp.end())
      hist_rdp[{addr, new_pc}] = 0;
    return hist_rdp[{addr, new_pc}];
    }
};

} // namespace replacement_policy
} // namespace gem5

#endif // __MEM_CACHE_REPLACEMENT_POLICIES_Belady_RP_HH__
```

Listing 3: Algorithm implementation

build کردن gem۵

برای Build کردن gem۵ از کامند زیر بهره می بریم

```
    scons build/X86/gem5.opt -j 9
```

Listing 4: cloning gem5

اضافه کردن آپشن replacement policy

این مورد نیز از اطلاعات تمرین ۱۰ام استفاده می کنیم.

برای اضافه کردن قابلیت سیاست جایگزینی[11] هنگام اجرای شبیه‌سازی مراحل زیر را اجرا کنید. ابتدا در فایل موجود در مسیر configs/common/ObjectList خط زیر را اضافه کنید:

```
        repl_list = ObjectList(getattr(m5.objects, 'BaseReplacementPolicy',
           None))
```

سپس قابلیت‌های زیر را به عنوان قابلیت‌های شبیه‌سازی، در فایل configs/common/Options.py به تابع addNoISAOptions اضافه کنید:

```
        parser.add_argument("--l2_repl", default="LRURP",
        choices=ObjectList.repl_list.get_names(),
        help = "replacement policy for l2")
```

نهایتا سه خط زیر را به انتهای تابع __get__cache__opts در فایل موجود در مسیر configs/common/CacheConfig.py اضافه کنید:

```
        replacement_policy_attr = f"{level}_repl"
        if hasattr(options, replacement_policy_attr):
        opts["replacement_policy"] = ObjectList.repl_list.get(getattr(options,
           replacement_policy_attr))()
```

حال می‌توانید با استفاده از قابلیت l2__repl– نوع سیاست جایگزینی را هنگام شبیه‌سازی برای حافظه نهان لایه دوم تعیین کنید.

طرز استفاده

پس از انجام تمام کار های ذکر شده می توان برای استفاده از policy جدید، از فرمت دستور زیر استفاده کرد.

```
./build/X86/gem5.opt configs/deprecated/example/se.py -c [binary file]
--caches --l2cache --l2\_size=4kB --mem-type=DDR4\_2400\_16x4
--cacheline\_size 128 --l2\_repl=BeladyRP
```

Listing 5: cloning gem5

پیاده‌سازی برنامه‌های محک و تست کردن سیاست گفته‌شده

در ابتدا به ۱۰ تست‌بنچ طراحی‌شده و دلیل وجود هر یک می‌پردازیم.

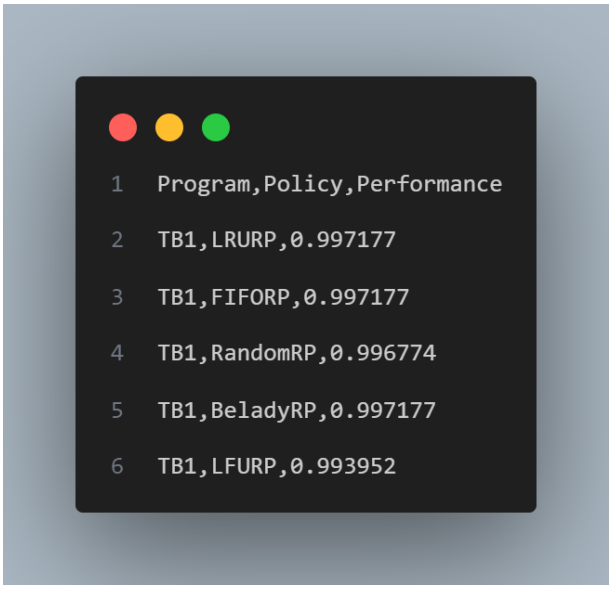
پس از قطعه کد هر برنامه‌ی محک می‌توانید نتایج به دست‌آمده از اعمال ۵ سیاست متفاوت روی آن را مشاهده کنید. لازم به ذکر است که سیاست مربوط به پروژه همان BeladyRP است.

برنامه‌ی محک اول

این برنامه‌ی محک به تست کردن دسترسی‌های متوالی (Sequential Access) می‌پردازد. در این برنامه در یک آرایه مجموع اعضای آرایه اصلی با شروع از عضو اول ذخیره شده‌است.

```c
#include <stdio.h>
#define ARRAY_SIZE 5000

int main() {
    int array[ARRAY_SIZE];
    int cumulative_sum[ARRAY_SIZE];

    for (int i = 0; i < ARRAY_SIZE; i++) {
        array[i] = i;
    }
    cumulative_sum[0] = array[0];

    for (int i = 1; i < ARRAY_SIZE; i++) {
        cumulative_sum[i] = cumulative_sum[i - 1] + array[i];
    }

    return 0;
}
```

Listing 6: TestBench1



```
Program,Policy,Performance
TB1,LRURP,0.997177
TB1,FIFORP,0.997177
TB1,RandomRP,0.996774
TB1,BeladyRP,0.997177
TB1,LFURP,0.993952
```

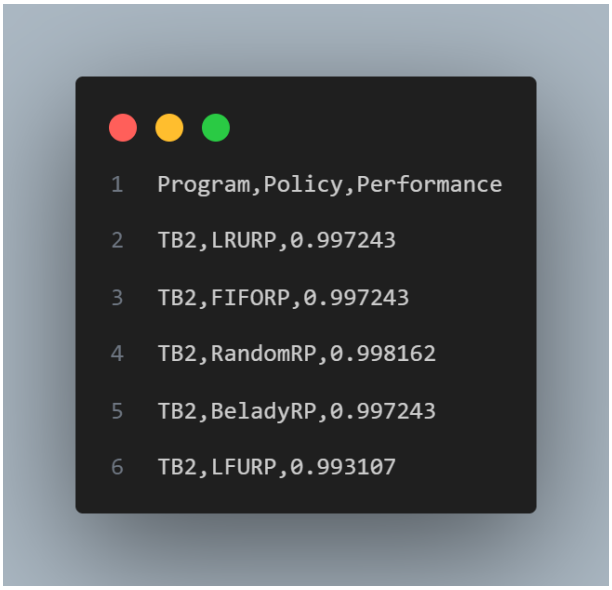
برنامه‌ی محک دوم

این برنامه‌ی محک به تست کردن دسترسی‌های تصادفی (Random Access) می‌پردازد؛ این برنامه پس از مقداردهی خانه‌های یک آرایه شروع به خواندن مقدار خانه‌ها به صورت تصادفی می‌کند.

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define ARRAY_SIZE 5000

int main() {
    int array[ARRAY_SIZE];
    srand(time(NULL));

    for (int i = 0; i < ARRAY_SIZE; i++) {
        array[i] = i;
    }

    for (int i = 0; i < ARRAY_SIZE; i++) {
        int index = rand() % ARRAY_SIZE;
        int value = array[index];
    }

```

```
    return 0;
}
```

Listing 7: TestBench2



```
Program,Policy,Performance
TB2,LRURP,0.997243
TB2,FIFORP,0.997243
TB2,RandomRP,0.998162
TB2,BeladyRP,0.997243
TB2,LFURP,0.993107
```

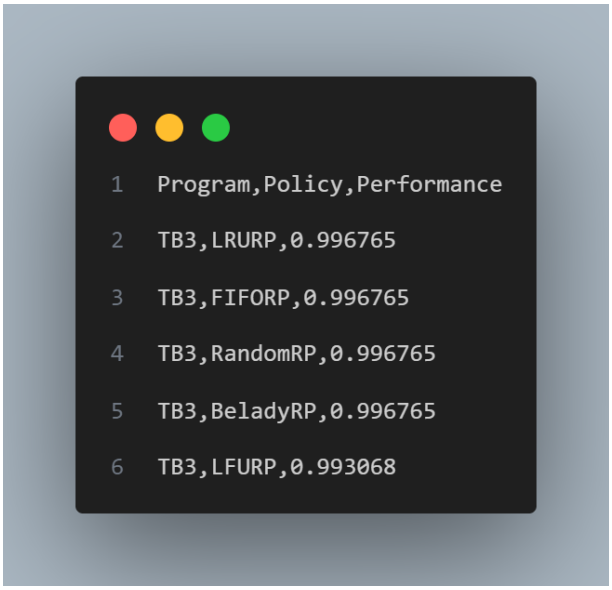
برنامه‌ی محک سوم

این برنامه‌ی محک به تست کردن دسترسی‌ها به صورت منظم می‌پردازد؛ به این صورت که مقدار خانه‌ها را بافاصله‌ای ثابت می‌خواند؛ سپس به ابتدا برگشته، یک خانه جلو می‌رود و این مراحل را تکرار می‌کند.

```c
#include <stdio.h>
#define ARRAY_SIZE 5000
#define LOCALITY_SIZE 50

int main() {
    int array[ARRAY_SIZE];

    for (int i = 0; i < ARRAY_SIZE; i++) {
        array[i] = i;
    }

    for (int start = 0; start < LOCALITY_SIZE; start++) {
```

```
        for (int i = start; i < ARRAY_SIZE; i += LOCALITY_SIZE) {
            int value = array[i];
        }
    }

    return 0;
}
```

Listing 8: TestBench3



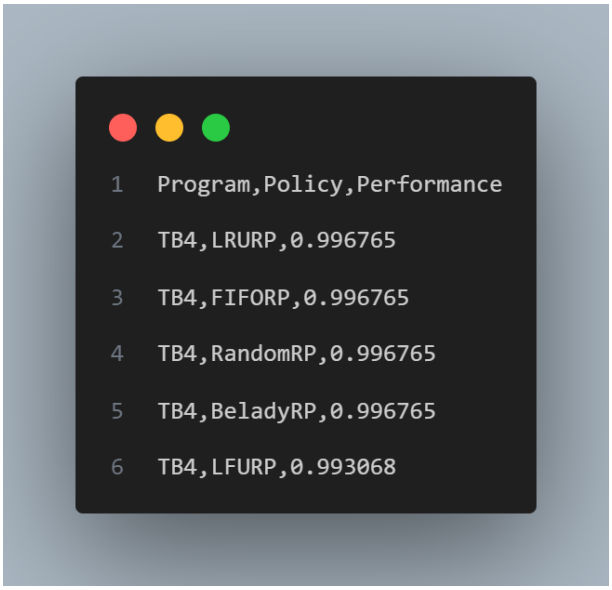
برنامه‌ی محک چهارم

این برنامه‌ی محک مشابه قبلی است، با این تفاوت که تکرار از ابتدا ندارد و یک بار به صورت منظم و با فاصله‌ی ثابت مقدار خانه آرایه را می‌خواند. در واقع این بنامه‌ی محک به تست کردن Strided Access می‌پردازد.

```
#include <stdio.h>
#define ARRAY_SIZE 5000
#define STRIDE 4

int main() {
    int array[ARRAY_SIZE];

```

```
    for (int i = 0; i < ARRAY_SIZE; i++) {
        array[i] = i;
    }

    for (int i = 0; i < ARRAY_SIZE; i += STRIDE) {
        int value = array[i];
    }

    return 0;
}
```

Listing 9: TestBench4



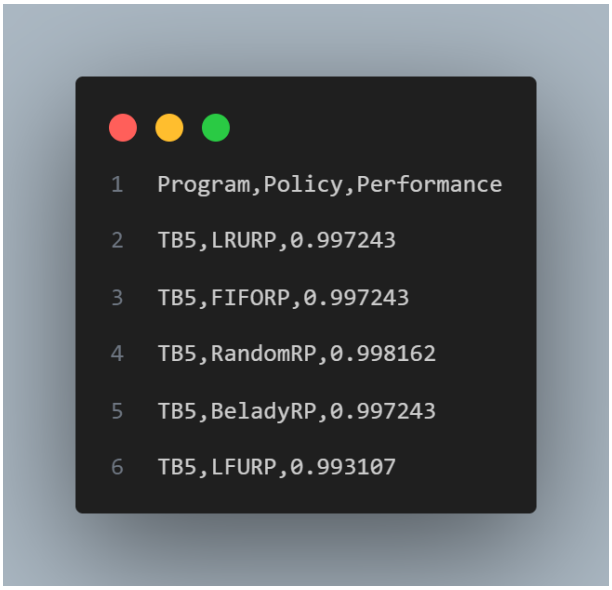
برنامه‌ی محک پنجم

این برنامه‌ی محک به تست کردن دسترسی‌ها به صورت نامنظم ولی رو به جلو می‌پردازد؛ به این صورت که با شروع از خانه‌ی اول، به تعداد تصادفی و محدودی از خانه‌ها جلوتر می‌رود و این روند را تکرار می‌کند.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define ARRAY_SIZE 5000
```

```
#define MAX_STRIDE 7

int main() {
    int array[ARRAY_SIZE];
    srand(time(NULL));

    for (int i = 0; i < ARRAY_SIZE; i++) {
        array[i] = i;
    }

    int random_stride = 1;

    for (int i = 0; i < ARRAY_SIZE; i += random_stride) {
        int value = array[i];
        random_stride = (rand() % MAX_STRIDE) + 1;
    }

    return 0;
}
```

Listing 10: TestBench5

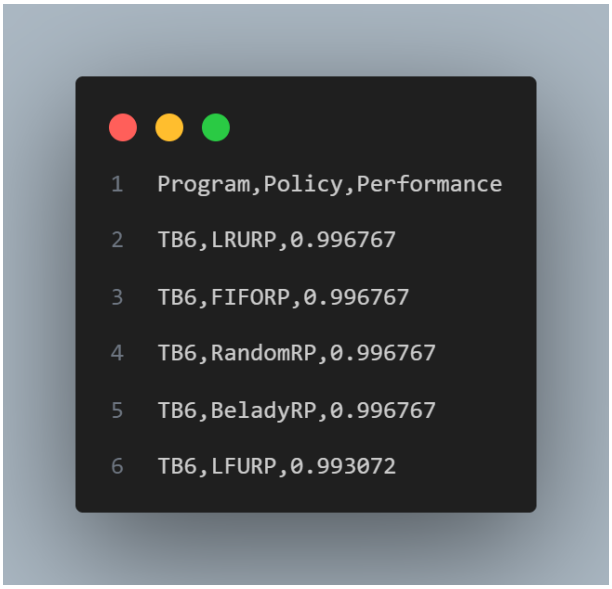


برنامه‌ی محک ششم

این برنامه‌ی محک به تست کردن دسترسی‌ها به صورت روبه‌جلو و با نظمی مشخص می‌پردازد؛ به ین صورت که با شروع از خانه‌ی اول، شمارنده هر بار در عددی ثابت ضرب می‌شود و آدرس خانه‌ی بعدی را ایجاد می‌کند.

```c
#include <stdio.h>
#define ARRAY_SIZE 5000

void access_pattern(int array[], int size) {
    for (int i = 0; i < size; i++) {
        int value = array[i];
    }
}

int main() {
    int array[ARRAY_SIZE];

    for (int i = 0; i < ARRAY_SIZE; i++) {
        array[i] = i;
    }

    for (int size = 50; size <= ARRAY_SIZE; size *= 10) {
        access_pattern(array, size);
    }

    return 0;
}
```

Listing 11: TestBench6



```
Program,Policy,Performance
TB6,LRURP,0.996767
TB6,FIFORP,0.996767
TB6,RandomRP,0.996767
TB6,BeladyRP,0.996767
TB6,LFURP,0.993072
```

برنامه‌ی محک هفتم

این برنامه‌ی محک مقدار تابع فیوناچی را به ازای ورودی ۴۸ محاسبه می‌کند. تابع فیبوناچی در این برنامه با استفاده از برنامه‌نویسی پویا نوشته شده‌است و دسترسی‌ها به حافظه‌ی موردنظر را تست می‌کند که به صورت بازگشتی و مطاب رابطه‌ی فیوناچی است.

```c
#include <stdio.h>
#define ARRAY_SIZE 50

int fib(int n, int memory[]) {
    if (n == 0 || n == 1)
        return 1;

    int x, y;
    if (memory[n - 1] != -1)
        x = memory[n - 1];
    else {
        x = fib(n - 1, memory);
        memory[n - 1] = x;
    }

    if (memory[n - 2] != -1)
        y = memory[n - 2];
```

```
    else {
        y = fib(n - 2, memory);
        memory[n - 2] = y;
    }

    return x + y;
}

int main() {
    int fibMemory[ARRAY_SIZE];

    for (int i = 0; i < ARRAY_SIZE; i++) {
        fibMemory[i] = -1;
    }

    fibMemory[0] = 1;
    fibMemory[1] = 1;

    fib(48, fibMemory);
}
```

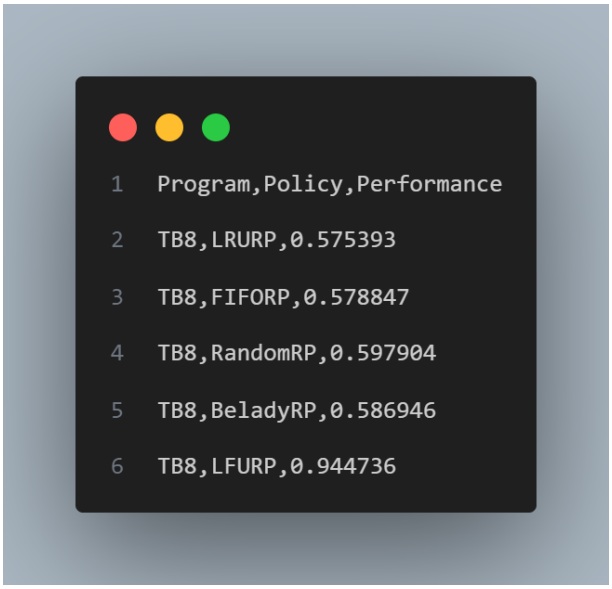
Listing 12: TestBench7

برنامه‌ی محک هشتم

این برنامه‌ی محک ابتدا به ترتیب خانه‌های سطرهای متوالی از دو ماتریس (آرایه‌ی دو بعدی) را مقداردهی می‌کند؛ سپس به محاسبه‌ی حاصل ضرب ماتریسی آن دو می‌پردازد و دسترسی به خانه‌های مختلف ماتریس‌ها را مورد تست قرار می‌دهد

```c
#include <stdio.h>
#define N 100

void matrix_multiplication(int mat1[N][N], int mat2[N][N], int result[N][N]) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            result[i][j] = 0;
            for (int k = 0; k < N; k++) {
                result[i][j] += mat1[i][k] * mat2[k][j];
            }
        }
    }
}

int main() {
    int matrix1[N][N], matrix2[N][N], result[N][N];

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            matrix1[i][j] = i + j;
            matrix2[i][j] = i - j;
        }
    }

    matrix_multiplication(matrix1, matrix2, result);
    return 0;
}
```

Listing 13: TestBench8



```
Program,Policy,Performance
TB8,LRURP,0.575393
TB8,FIFORP,0.578847
TB8,RandomRP,0.597904
TB8,BeladyRP,0.586946
TB8,LFURP,0.944736
```

برنامه‌ی محک نهم

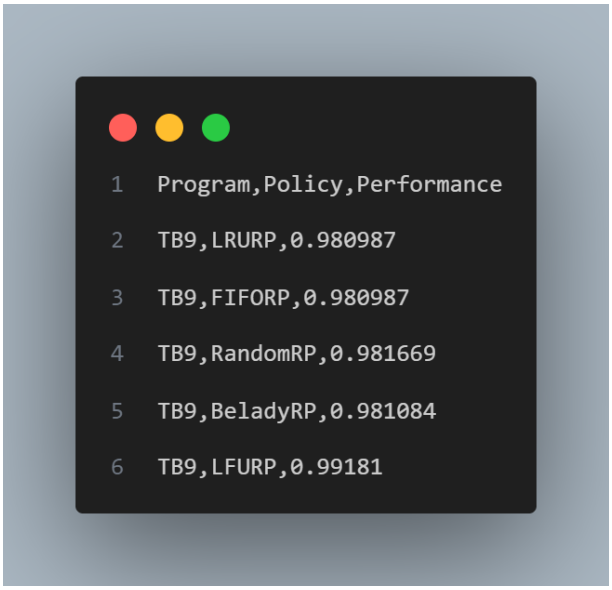
این برنامه‌ی محک به ساخت یک درخت باینری و سپس پیمایش میان‌ترتیب آن می‌پردازد و دسترسی‌ها به خانه‌های مختلف آرایه‌ای که درخت را تشکیل می‌دهد، را مورد بررسی قرار می‌دهد.

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct Node {
    int data;
    struct Node *left;
    struct Node *right;
} Node;

Node *newNode(int data) {
    Node *node = (Node *)malloc(sizeof(Node));
    node->data = data;
    node->left = NULL;
    node->right = NULL;
    return node;
}

Node *insertLevelOrder(int arr[], Node *root, int i, int n) {
```

```
    if (i < n) {
        Node *temp = newNode(arr[i]);
        root = temp;

        root->left = insertLevelOrder(arr, root->left, 2 * i + 1, n);
        root->right = insertLevelOrder(arr, root->right, 2 * i + 2, n);
    }
    return root;
}

void inorder(Node *root) {
    if (root != NULL) {
        inorder(root->left);
        int value = root->data;
        inorder(root->right);
    }
}

void freeTree(Node *root) {
    if (root != NULL) {
        freeTree(root->left);
        freeTree(root->right);
        free(root);
    }
}

int main() {
    int numNodes = 5000;
    int *arr = (int *)malloc(numNodes * sizeof(int));
    for (int i = 0; i < numNodes; i++) {
        arr[i] = i + 1;
    }

    Node *root = insertLevelOrder(arr, root, 0, numNodes);
    inorder(root);
    freeTree(root);
```

```
    free(arr);
    return 0;
}
```

Listing 14: TestBench9



```
Program,Policy,Performance
TB9,LRURP,0.980987
TB9,FIFORP,0.980987
TB9,RandomRP,0.981669
TB9,BeladyRP,0.981084
TB9,LFURP,0.99181
```

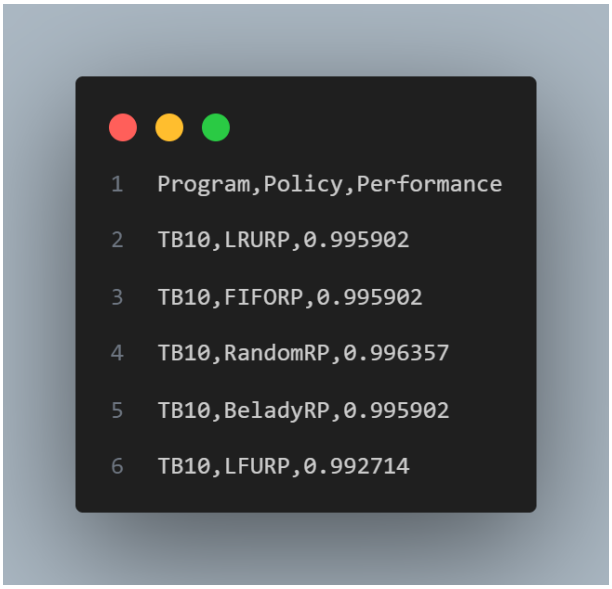
برنامه‌ی محک دهم

این برناه‌ی محک به مرتب‌سازی سریع یک آرایه می‌پردازد و دسترسی‌ها به خانه‌های متعدد این آرایه را مورد تست قرار می‌دهد.

```
#include <stdio.h>
#include <stdlib.h>
#define ARRAY_SIZE 5000

void swap(int *a, int *b) {
    int t = *a;
    *a = *b;
    *b = t;
}

int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = (low - 1);
```

```
14
15      for (int j = low; j <= high - 1; j++) {
16          if (arr[j] < pivot) {
17              i++;
18              swap(&arr[i], &arr[j]);
19          }
20      }
21      swap(&arr[i + 1], &arr[high]);
22      return (i + 1);
23  }
24
25  void quickSort(int arr[], int low, int high) {
26      if (low < high) {
27          int pi = partition(arr, low, high);
28          quickSort(arr, low, pi - 1);
29          quickSort(arr, pi + 1, high);
30      }
31  }
32
33  int main() {
34      int arr[ARRAY_SIZE];
35      for (int i = 0; i < ARRAY_SIZE; i++) {
36          arr[i] = rand() % ARRAY_SIZE;
37      }
38
39      quickSort(arr, 0, ARRAY_SIZE - 1);
40      return 0;
41  }
```

Listing 15: TestBench10


Program,Policy,Performance
TB10,LRURP,0.995902
TB10,FIFORP,0.995902
TB10,RandomRP,0.996357
TB10,BeladyRP,0.995902
TB10,LFURP,0.992714

مقایسه‌ی سیاست‌های مختلف

یک کد پایتون به صورت زیر نوشتیم که ده برنامه محک مختلف را روی این سیاست و چهار سیاست معروف شبیه سازی کند و نتایج شبیه سازی را در قالب چند نمودار و یک فایل با فرمت csv ارائه دهد. در ادامه به شرح نحوه‌ی عملکرد این کد می‌پردازیم.

```
1
2  import subprocess
3  import pandas as pd
4  import matplotlib.pyplot as plt
5  import os
6
7  def run_gem5_simulation(program, policy, results):
8      command = [
9          "./build/X86/gem5.opt",
10         "configs/deprecated/example/se.py",
11         "-c", f"binary/{program}",
12         "--caches",
13         "--l2cache",
14         "--l2_size=4kB",
15         "--mem-type=DDR4_2400_16x4",
16         "--cacheline_size", "64",
17         f"--l2_repl={policy}"
18     ]
19
20     try:
21         subprocess.check_output(command, universal_newlines=True)
22
23         performance = extract_performance_metric('m5out/stats.txt')
24
25         if performance is not None:
26             results.append((program, policy, performance))
27     except subprocess.CalledProcessError as e:
28         print(f"Error running {program} with policy {policy}: {e}")
29
30 def extract_performance_metric(stats_file):
```

```
    try:
        with open(stats_file, 'r') as f:
            for line in f:
                if 'l2.overallMissRate::cpu.data' in line:
                    return float(line.split()[1])
    except Exception as e:
        print(f"Error reading {stats_file}: {e}")
    return None

def run_simulations(programs, policies):
    results = []

    for program in programs:
        for policy in policies:
            run_gem5_simulation(program, policy, results)

    return results

def generate_results_csv_and_plots(results, filename):
    df = pd.DataFrame(results, columns=['Program', 'Policy', 'Performance'
    ])
    df = df.dropna(subset=['Performance'])

    df.to_csv(filename, index=False)

    for program in df['Program'].unique():
        plt.figure()
        subset = df[df['Program'] == program]
        plt.bar(subset['Policy'], subset['Performance'])
        plt.xlabel('Policy')
        plt.ylabel('Performance')
        plt.title(f'Performance of {program} with different policies')
        plt.savefig(f'{program}_performance.png')

# programs = [f'bench{i}' for i in range(1, 11)]
programs = ['bench1']
```

```
# policies = ['Belady', 'LRU', 'FIFO', 'LFU', 'RANDOM']
policies = ['LRURP', 'FIFORP', 'RandomRP']

results = run_simulations(programs, policies)
print(results)
generate_results_csv_and_plots(results, 'simulation_results.csv')
```

Listing 16: Comparing Script implementation

ابتدا قالب دستور مورد نظرمان را آماده کردیم. حال کافیست به این قالب برنامه‌ی محک و سیاست مدنظرمان را به عنوان پارامترهای ورودی آن بدهیم تا فرآیند شبیه‌سازی انجام شود.

```

command = [
        "./build/X86/gem5.opt",
        "configs/deprecated/example/se.py",
        "-c", f"binary/{program}",
        "--caches",
        "--l2cache",
        "--l2_size=4kB",
        "--mem-type=DDR4_2400_16x4",
        "--cacheline_size", "64",
        f"--l2_repl={policy}"
    ]
```

Listing 17: Command

حال، Gem۵ خروجی مدنظر ما را در فایل stats ذخیره می‌کند. خط مورد نظر ما در این فایل، شامل عبارت زیر است:

```
l2.overallMissRate::cpu.data
```

حال، با استفاده از اجرای حلقه بر روی لیست برنامه‌های محک و سیاست‌های مدنظر، شبیه‌سازی را برای تمامی حالت‌های ممکن اجرا می‌کنیم.

```
    for program in programs:
        for policy in policies:
            run_gem5_simulation(program, policy, results)
```

Listing 18: Running all simulations

در نهایت، داده‌های به دست آمده از شبیه‌سازی‌ها را plot می‌کنیم و به ازای هر برنامه محک یکسان، rate miss سیاست‌های مختلف را مقایسه می‌کنیم.

نمونه‌ای از خروجی‌های به دست آمده، در تصویر زیر قابل مشاهده است. البته به دلیل بسیار جزئی بودن تفاوت‌ها، ، می‌توانیم تغییری روی هایplot ایجاد شده اعمال کنیم که تفاوتشان نمایان‌تر باشد.

برای اجرای این اسکریپت به صورت موازی نیز تغییرات زیر را اعمال کردیم:

```
    output_dir = f"m5out_{program}_{policy}"
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    command = [
        "./build/X86/gem5.opt",
        "configs/deprecated/example/se.py",
        "-c", f"binary/{program}",
        "--caches",
        "--l2cache",
        "--l2_size=4kB",
        "--mem-type=DDR4_2400_16x4",
        "--cacheline_size", "128",
        f"--l2_repl={policy}",
        "--stats-file", f"{output_dir}/stats.txt"
    ]
```

Listing 19: Multi core version of command

دلیل اعمال تغییرات گفته شده نیز آن بود که اگر قرار باشد تمامی هسته‌ها خروجی‌شان را بر روی یک فایل یکسان stats ذخیره کنند، ممکن است به عنوان مثال هنگامی که ترد اول می‌خواهد نتایجش را از فایل بخواند، ترد دوم نتایج را تغییر داده باشد و در نتیجه خروجی ما اشتباه شود. لذا برای هر یک از شبیه‌سازی‌ها، یک دایرکتوری متفاوت در نظر گرفتیم که از این مشکل جلوگیری شود.

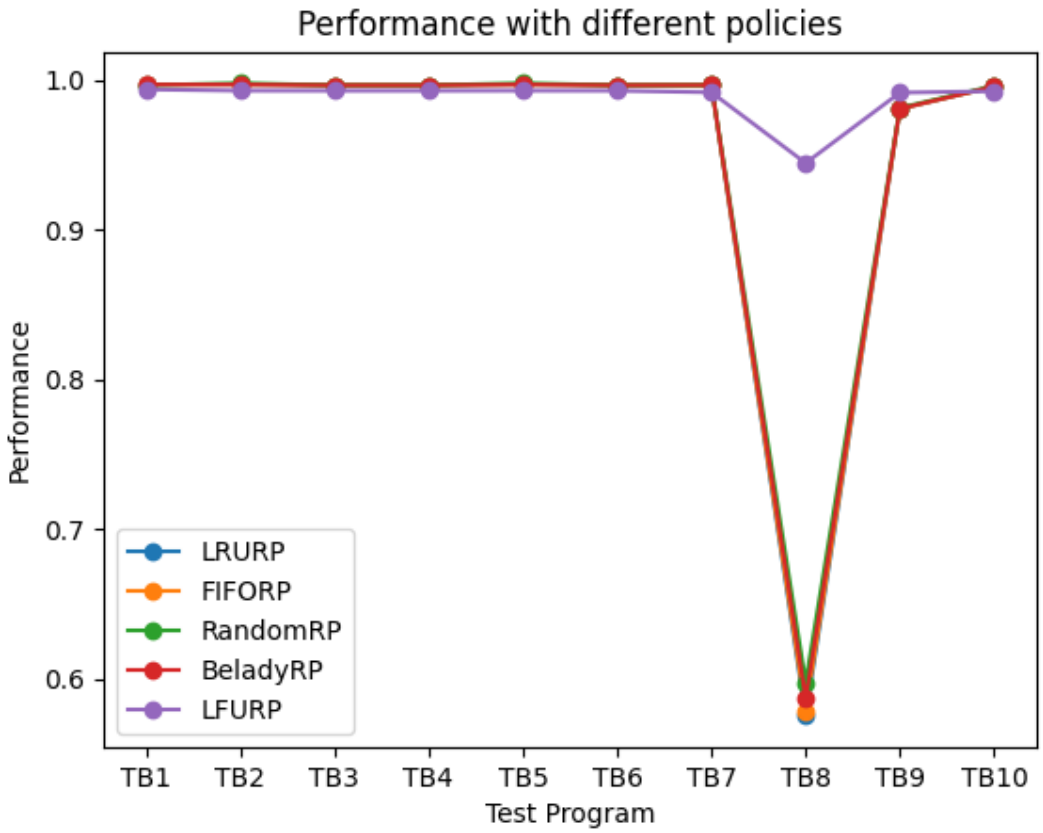
قسمتی از کد که تحت ویرایش قرار گرفت، در ادامه آمده است. البته لازم به ذکر است که در نهایت اجرای این حالت موفقیت آمیز نبود زیرا فایل‌های stats مربوطه ساخته نشدند.

```
import multiprocessing as mp
```

```python
import subprocess
import pandas as pd
import matplotlib.pyplot as plt
import os

def run_gem5_simulation(program, policy, results):
    output_dir = f"m5out_{program}_{policy}"
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    command = [
        "./build/X86/gem5.opt",
        "configs/deprecated/example/se.py",
        "-c", f"binary/{program}",
        "--caches",
        "--l2cache",
        "--l2_size=4kB",
        "--mem-type=DDR4_2400_16x4",
        "--cacheline_size", "128",
        f"--l2_repl={policy}",
        "--stats-file", f"{output_dir}/stats.txt"
    ]

    try:

        subprocess.check_output(command, universal_newlines=True)

        stats_file = os.path.join(output_dir, 'stats.txt')
        performance = extract_performance_metric(stats_file)

        if performance is not None:
            results.append((program, policy, performance))
    except subprocess.CalledProcessError as e:
        print(f"Error running {program} with policy {policy}: {e}")

def extract_performance_metric(stats_file):
```

```
    try:
        with open(stats_file, 'r') as f:
            for line in f:
                if 'l2.overallMissRate::cpu.data' in line:
                    return float(line.split()[1])
    except Exception as e:
        print(f"Error reading {stats_file}: {e}")
    return None

def run_simulations(programs, policies):
    manager = mp.Manager()
    results = manager.list()
    jobs = []

    for program in programs:
        for policy in policies:
            p = mp.Process(target=run_gem5_simulation, args=(program,
   policy, results))
            jobs.append(p)
            p.start()

    for job in jobs:
        job.join()

    return list(results)
```

Listing 20: Multi core comparing Script implementation


Performance with different policies
Performance
1.0
0.9
0.8
0.7
0.6
LRURP
FIFORP
RandomRP
BeladyRP
LFURP
TB1
TB2
TB3
TB4
TB5
TB6
TB7
TB8
TB9
TB10
Test Program