برای بخش اول این دایرکتوریها و فایل کانفیگ را مطالعه کنید و به موارد زیر پاسخ دهید.

۱.ساب دایرکتوریهای فوق چه کاری را انجام میدهند و چه کاربردی دارند؟ به طور خالصه پاسخ دهید.

src

برنامه های اصلی چمپسیم در این بخش قرار دارند یعنی این کد ها باعث میشوند برنامه با تنظیمات متفاوت کار کند.

Prefetcher

پریفچ کردن به این معناست که دستورات یا اطلاعات به کش اورده شوند قبل از اینکه پردازنده به آنها نیاز داشته باشد. اینگونه زمان کمتری برده میشود. الگوریتم های پریفچ مختلفی در این فولدر وجود دارند همچنین میتوانیم یکی بسازیم و از آن استفاده کنیم

Placement

در کش، وقتی بلوک جدیدی را میخواهیم قرار دهیم، باید بر حسبی انتخاب کنیم در جای کدام بلوک قرار بگیرد. درون این فولدر الگوریتم های مختلف جایگذاری وجود دارد، همچنین خودمان میتوانیم تعریف کنیم.

Tracer

دستوراتی که چمپسیم اجرا میکند، به شکل trace هستند چون کش چمپسیم براساس trace ها کار میکند به این معنا که با استفاده از الگوی دسترسی به دستورات، پیشبینی میکند چه دستوراتی در آینده نیاز خواهند شد. در این فولدر، وسایل مورد نیاز برای ساخت یک trace در این فولدر قرار داده شده است.

Champsim_config.json

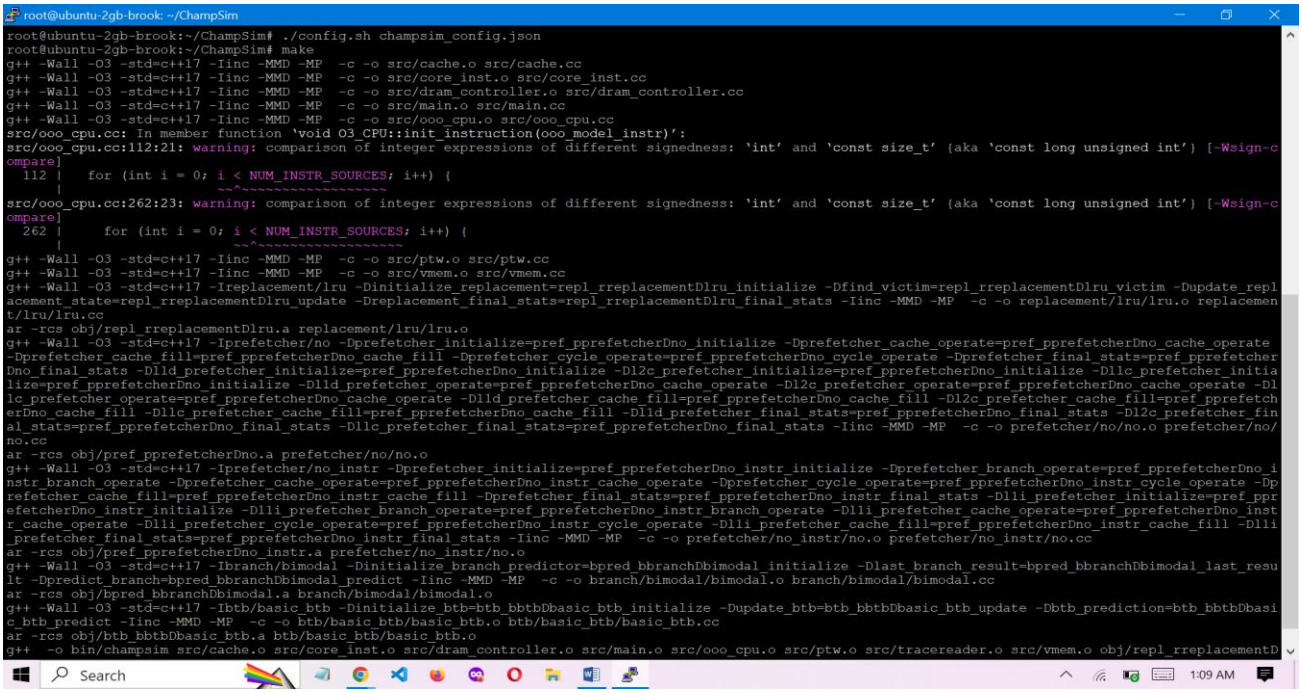
در این فایل همه ی تنظیمات قرار داده شده اند. مثلا اندازه ی مموری، مشخصات cpu و مشخصات کش ها و نحوه ی عملکردشان. این مشخصات را بسته به نیاز میتوان تغییر داد.

۲. فایل json.config_champsim چه مواردی را مقدار ردهی میکند. در کدام بخش از تنظیمات میتوان سیاسیت جایگزینی را مشخص کرد؟

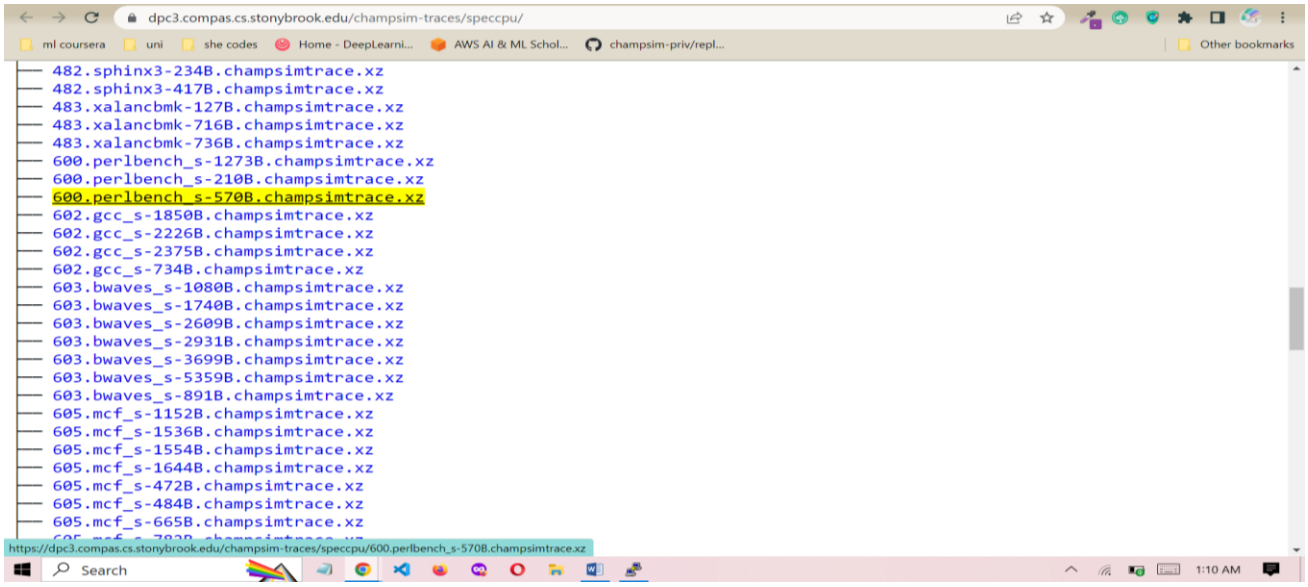
سایز هر word, page را مشخص میکند. فرکانس، ظرفیت فچ و دیکود و اجرا و الگوریتم پیشبینی برنچ که همه مربوط به cpu است. تعداد set,way و تنظیمات پریفچ کش سطح یک و دو و اخر را مشخص میکند. همچنین مشخصات مربوط به مموری فیزیکی مانند فرکانس، ردیف،ستون و... را مشخص میکند.

در خط ۱۵۸، در مشخصات مربوط به اخرین مرحله ی کش llc سیاست جایگزینیreplacement مشخص میشود.

./config.sh champsim_config.json , make



choose a trace: https://dpc3.compas.cs.stonybrook.edu/champsim-traces/speccpu/600.perlbench_s-570B.champsimtrace.xz



4. Run a Simulation:

bin/champsim --warmup_instructions 100000000 --simulation_instructions 300000000 https://dpc3.compas.cs.stonybrook.edu/champsim-traces/speccpu/600.perlbench_s-570B.champsimtrace.xz

عملکرد LRU را بعدا همراه با روش های دیگر تحلیل میکنیم.

فاز دوم

با انجام دادن مراحل پیشرو یکی از سیاستهای جایگزاری FIFO, FILO, MRU را به انتخاب خود پیادهسازی کنید، مراحل پیاده سازی را توضیح دهید، کامپایل و تست کنید و در نهایت نتایج را با روش LRUمقایسه کنید و گزارش دهید.

MRU

mkdir replacement/mru

cp replacement/lru/lru.cc replacement/mru/mru.cc

میخواهیم روش MRU را پیاده سازی کنیم. تنها تفاوت کد این روش با LRU این است:

LRU:

```
// find replacement victim
uint32_t CACHE::find_victim(uint32_t cpu, uint64_t instr_id, uint32_t
set, const BLOCK* current_set, uint64_t ip, uint64_t full_addr, uint32_t
type)
{
  // baseline MRU
  return std::distance(current_set, std::max_element(current_set,
std::next(current_set, NUM_WAY), lru_comparator<BLOCK, BLOCK>()));
}
```

MRU:

```
// find replacement victim
uint32_t CACHE::find_victim(uint32_t cpu, uint64_t instr_id, uint32_t
set, const BLOCK* current_set, uint64_t ip, uint64_t full_addr, uint32_t
type)
{
  // baseline MRU
  return std::distance(current_set, std::min_element(current_set,
std::next(current_set, NUM_WAY), lru_comparator<BLOCK, BLOCK>()));
}
```

هنگام پیدا کردن بلوکی که باید تخلیه شود، در lru مسن ترین انتخاب میشود و در mru جدیدترین.

حال champsim_config.json را نیز تغییر میدهیم.

LLC {replacement = mru}

حال دوباره باید کامپایل کنیم چون کانفیگوریشن فایل متفاوت شده:

./config.sh champsim_config.json , make

حال برنامه را با تنظیمات قبلی، دوباره اجرا میکنیم:

bin/champsim --warmup_instructions 100000000 --simulation_instructions 300000000 https://dpc3.compas.cs.stonybrook.edu/champsim-traces/speccpu/600.perlbench_s-570B.champsimtrace.xz

---

FIFO

mkdir replacement/fifo

cp replacement/lru/lru.cc replacement/fifo/fifo.cc

میخواهیم روش FIFO را پیاده سازی کنیم. تفاوت کد این روش با LRU این است:

LRU:

```
// called on every cache hit and cache fill
void CACHE::update_replacement_state(uint32_t cpu, uint32_t set, uint32_t
way, uint64_t full_addr, uint64_t ip, uint64_t victim_addr, uint32_t
type, uint8_t hit)
{
  if (hit && type == WRITEBACK)
    return;
  auto begin = std::next(block.begin(), set * NUM_WAY);
  auto end = std::next(begin, NUM_WAY);
  uint32_t hit_lru = std::next(begin, way)->lru;
  std::for_each(begin, end, [hit_lru](BLOCK& x) {
    if (x.lru <= hit_lru)
      x.lru++;
  });
  std::next(begin, way)->lru = 0; // promote to the MRU position
}
```

FIFO:

```
// called on every cache hit and cache fill
void CACHE::update_replacement_state(uint32_t cpu, uint32_t set, uint32_t
way, uint64_t full_addr, uint64_t ip, uint64_t victim_addr, uint32_t
type,uint8_t hit){
  // on cache fill
  if (not hit) {
    auto begin = std::next(block.begin(), set * NUM_WAY);
    auto end = std::next(begin, NUM_WAY);
    uint32_t hit_lru = std::next(begin, way)->lru;
    std::for_each(begin, end, [hit_lru](BLOCK& x) { x.lru++; });
    std::next(begin, way)->lru = 0; // promote to the first position
  }
}
```

برای اینکار، هنگام جایگذاری بلوک جدید، بلوک آخر انتخاب میشود. سپس سن بلوک پس از جایگزینی صفر میشود و به سن بقیه ی بلوک ها یکی اضافه میشود. اینگونه یک صف FIFO درست میشود.

حال champsim_config.json را نیز تغییر میدهیم.

LLC {replacement = fifo}

حال دوباره باید کامپایل کنیم چون کانفیگوریشن فایل متفاوت شده:

./config.sh champsim_config.json , make

حال برنامه را با تنظیمات قبلی، دوباره اجرا میکنیم:

bin/champsim --warmup_instructions 100000000 --simulation_instructions 300000000 https://dpc3.compas.cs.stonybrook.edu/champsim-traces/speccpu/600.perlbench_s-570B.champsimtrace.xz

عملکرد الگوریتم های جایگزینی متفاوت (FIFO MRU LRU)

با توجه به

۱. تغییر تنظیمات در بخش LLC فایل کانفیگوریشن انجام شده است.

۲. در بین نتایج، تنها تغییر محسوس در بخش LLC رخ داده است.

ما نتیجه میگیریم که تنها نتایج مربوط به LLC مهم است.

LLC results for LRU:

	access	Hit	miss
Total access	481437	177551	303886
Load access	381279	92887	288392
RFO access	22411	7749	14662
Writeback access	77627	76858	769
Translation access	120	57	63

LLC AVERAGE MISS LATENCY: 155.194 cycles

LLC results for MRU:

	access	Hit	miss
Total access	481240	192577	288663
Load access	381280	152003	229277
RFO access	22410	9134	13276
Writeback access	77430	31382	46048
Translation access	120	58	62

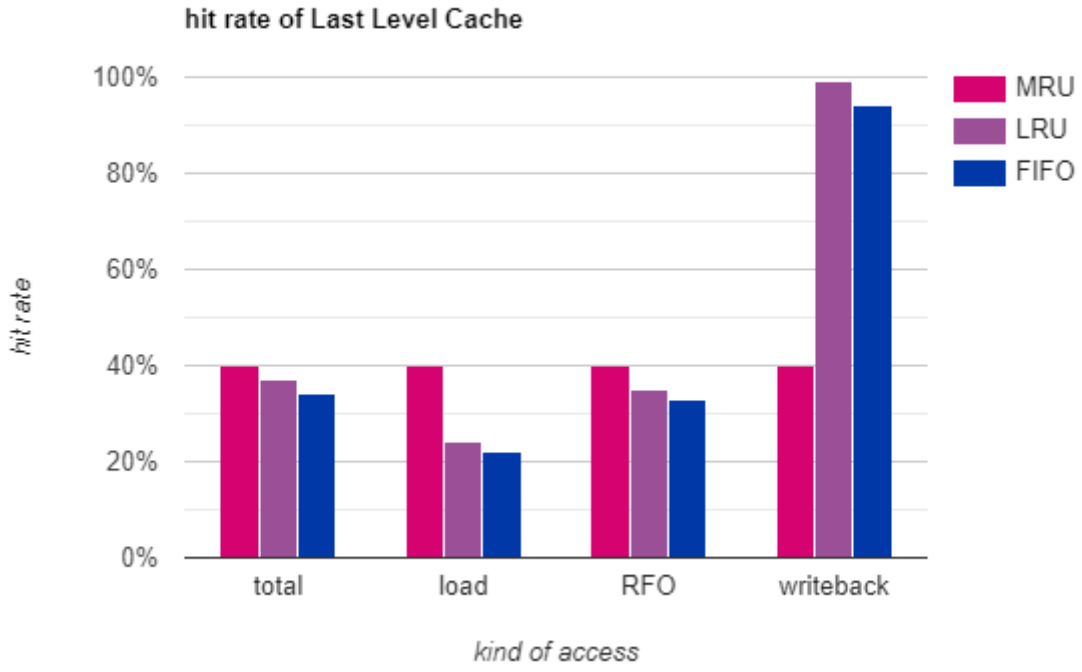
LLC AVERAGE MISS LATENCY: 132.344 cycles

LLC results for FIFO:

	access	Hit	miss
Total access	481433	163436	317997
Load access	381282	82621	298661
RFO access	22409	7474	14935
Writeback access	77622	73290	4332
Translation access	120	51	69

LLC AVERAGE MISS LATENCY: 154.032 cycles

بررسی کلی عملکرد الگوریتم ها بر اساس hit rate



به طور کلی MRU عملکرد بهتری دارد اما بزرگترین ضعف آن در writeback است که بسیار ضعیف تر از LRU و حتی FIFO عمل میکند. این ضعف در نظر من به این خاطر است که دیتایی گرفته میشود، سپس یک دیتای دیگر گرفته میشود که اگر میس بخورد، جای دیتای اول را در کش میگیرد. سپس تغییراتی که با استفاده از دیتای دوم بر اولی اعمال میشود. وقتی میخواهیم این آپدیت را روی دیتای اول write کنیم، با میس مواجه میشویم.

در برنامه هایی که write های زیادی دارد، بهتر است از LRU استفاده شود، در غیر این صورت، MRU بهتر عمل میکند. FIFO هم به طور کلی عملکرد ضعیف تری نسبت به LRU دارد در همه ی موارد.

همچنین شایسته ی توجه است که MRU در همه ی موارد یکسان(۴۰٪) عمل نموده در حالی که FIFO, LRU در writeback تفاوت چشمگیری با بقیه موارد دارند.