Advance Computer Architecture
Project 1: L2 Cache Replacement Policies

1. Not Recently Used (NRU)
   a) Requirement
      - RRPV is single-bit
      - Scanning starts from the head.
   b) Implementation
      - cache.h
        i. cache_blk_t add the member **RRPV** to store RRPV value of each block.
        ii. cache_t has member **RRPV_max**, **RRPV_hit**, **RRPV_new, nru_ptr**
            - **nru_ptr** indicate the index of the replacement block
            - **RRPV_max** means the max value of RRPV, which indicates this block need to be replaced.
            - **RRPV_hit** means RRPV value of re-referenced block
            - **RRPV_new** means RRPV value of the new-coming block
      - cache.c
        i. create_cache
            - Define RRPV parameter when creating cache
                - Single-bit version so that (**RRPV_max**, **RRPV_hit**, **RRPV_new**) = (1, 0, 1)

```
switch(policy){
    case NRU:    cp->RRPV_max = 1;
                 cp->RRPV_hit = 0;
                 cp->RRPV_new = 1;
                 break;
    default:     cp->RRPV_max = 0;
                 cp->RRPV_hit = 0;
                 cp->RRPV_new = 0;
                 break;
}
```

            - Initialize **nru_ptr** to 0 due to the requirement “Scanning starts from the head.”
            - Initialize RRPV value of each block

```
for (j = 0; j < assoc; j++) {
    /* locate next cache block */
    blk = CACHE_BINDEX(cp, cp->data, bindex);
    bindex++;

    /* invalidate new cache block */
    blk->RRPV   = cp->RRPV_max;
    blk->status = 0;
    blk->tag    = 0;
    blk->ready  = 0;
    blk->user_data = (usize != 0 ? (byte_t*)calloc(usize, sizeof(byte_t)) : NULL);
```

        ii. char2policy
            - Add case about NRU

iii. cache_access

- Add replacement index pointer **repl_ptr** to search the victim
  - Default value is 0 due to the requirement (scanning starts from the head.)
  - The pointer must be static type to preserve the value after this function returns.
- Add flag **victim** to indicates whether the victim is found, 1 means found
- Selection of victim block on cache miss
  - If RRPV value indicates replaced, then **set victim_find** and pointed by **repl**

```
/* select the appropriate block to replace, and re-link this entry to
   the appropriate place in the way list */
switch (cp->policy) {
    case LRU:
    case FIFO:
        repl = cp->sets[set].way_tail;
        update_way_list(&cp->sets[set], repl, Head);
        break;
    case Random:
        repl = CACHE_BINDEX(cp, cp->sets[set].blks, myrand()&(cp->assoc-1));
        break;
    case NRU:
        for( victim_find=0; !victim_find; cp->nru_ptr++){
            repl = CACHE_BINDEX(cp, cp->sets[set].blks, cp->nru_ptr%cp->assoc);
            if(repl->RRPV==cp->RRPV_max) victim_find = 1;
            else repl->RRPV++;
        }
        cp->nru_ptr = cp->nru_ptr % cp->assoc;
        break;
    default: panic("bogus replacement policy");
}
```

  - If its RRPV doesn't indicate replaced, increment RRPV value and continue to next block

- Set RRPV value of new-coming block on cache miss
  - RRPV value of new-coming block is defined as **RRPV_new** in cache

```
/* update block tags */
repl->tag = tag;
repl->status = CACHE_BLK_VALID;   /* dirty bit set on update */
switch(cp->policy){
    case NRU: repl->RRPV = cp->RRPV_new; break;
    default: break;
}

/* read data block */
lat += cp->blk_access_fn(Read, CACHE_BADDR(cp, addr), cp->bsize,
                         repl, now + lat);
```

- Set RRPV value of re-reference block on cache hit

```
/* if LRU replacement and this is not the first element of list, reorder */
switch(cp->policy){
    case LRU:
        /* move this block to head of the way (MRU) list */
        if (blk->way_prev) update_way_list(&cp->sets[set], blk, Head);
        break;
    case NRU: blk->RRPV = cp->RRPV_hit; break;
}
```

2. Dynamic Re-reference Interval Prediction on Set Dueling (DRRIP)
   a) Requirement
      - Distribution of dedicated sets

> (b) There are 32 dedicated SRRIP sets and 32 dedicated BRRIP sets, along with the remaining sets which are followers. Please employ the following hash functions in your implementation for cache configurations.
>
> i. An SRRIP set iff (**cache_set_ID**) mod ($\frac{total\#ofsets}{32}$) == 0
>
> ii. A BRRIP set iff (**cache_set_ID+1**) mod ($\frac{total\#ofsets}{32}$) == 0
>
> iii. The rest are the follower sets when both of the above conditions fail.

      - 3-bit RRPV(re-reference prediction value) for each cache line
      - 5-bit BIRCTR non-saturation counter for BRRIP
        - Whenever BIPCTR equals to 0, insert the line with long re-reference interval
        - Otherwise, the line with distant re-reference interval.
      - 10-bit PSEL counter for the selection between SRRIP and BRRIP of follower sets.

   b) Implementation of SRRIP and BRRIP
      - cache.h is same as NRU
      - cache.c
        i. create_cache
           - Define RRPV parameter when creating cache
             - 3-bit RRPV so that (**RRPV_max**, **RRPV_hit**, **RRPV_new**) = (8, 0, 7)

```
switch(policy){
    case NRU:    cp->RRPV_max = 1; cp->RRPV_hit=0; cp->RRPV_new=1; break;
    case SRRIP:
    case BRRIP: cp->RRPV_max = 8; cp->RRPV_hit=0; cp->RRPV_new=7; break;
    default:    cp->RRPV_max = 0; cp->RRPV_hit=0; cp->RRPV_new=0; break;
}
```

           - Initialize RRPV value of each block
             - Same as Not-Recently-Used

        ii. char2policy
           - Add case about SRRIP and BRRIP

        iii. cache_access
           - Selection of victim block on cache miss
             - Same as Not-Recently-Used

- Set RRPV value of new-coming block on cache miss
  - RRPV value of new-coming block is like NRU policy, but there is more case
  - SRRIP policy have constant RRPV for new-coming block

```
/* update block tags */
repl->tag = tag;
repl->status = CACHE_BLK_VALID;   /* dirty bit set on update */
switch(cp->policy){
    case NRU:
    case SRRIP: repl->RRPV = cp->RRPV_new; break;
    case BRRIP:
        if(cp->birctr%32==0)    repl->RRPV = cp->RRPV_new-1;
        else                repl->RRPV = cp->RRPV_new;
        cp->birctr++;
        break;
    default: break;
}
```

  - BRRIP policy determine RRPV depends on **birctr**, which is 5-bit BIRCTR in requirement and INISCA 2007 paper

- Set RRPV value of re-reference block on cache hit
  - Same as Not-Recently-Used

c) Implementation of DRRIP

- cache.h
  i. cache_set_t has new member **policy**, which is the replacement policy of the set
  ii. cache_t has its own **birctr**
- cache.c
  i. create_cache
    - Define RRPV parameter when creating cache
      - Same as BRRIP
    - Initialize RRPV value of each block
      - Same as NRU and BRRIP
    - Initialize the replacement policy of each set
      - BRRIP sets
        - Dedicated BRRIP: **j%32==0**, set policy must be set to BRRIP
      - SRRIP sets
        - Dedicated SRRIP: **j%32==1**, set policy must be set to SRRIP
        - Follower sets: The other cases, set policy is default SRRIP

```
/* slice up the data blocks */
for (bindex = 0, i = 0; i < nsets; i++) {

    if(policy==DRRIP){
        for(j=0; j<nsets; j++){
            if(j%(cp->nsets/32)==0) cp->sets[j].policy = BRRIP;
            else cp->sets[j].policy = SRRIP;
        }
    }
```

  ii. char2policy
    - Add case about DRRIP

iii. cache_access

- Add policy selection counter **psel** to determine replacement policy of follower sets
- Selection of victim block on cache miss
  - Same as Not-Recently-Used and BRRIP

- Set RRPV value of new-coming block on cache miss
  - If cache policy is DRRIP, the replacement policy is determined by set policy
  - **psel** is determined by **maxima()** or **minima()** to prevent overflow, and the overflow value is given by 10-bit PSEL counter in requirement.

```
/* update block tags */
repl->tag = tag;
repl->status = CACHE_BLK_VALID;   /* dirty bit set on update */
switch(cp->policy){
    case NRU:
    case SRRIP: repl->RRPV = cp->RRPV_new; break;
    case BRRIP:
        if(cp->birctr%32==0)    repl->RRPV = cp->RRPV_new-1;
        else                repl->RRPV = cp->RRPV_new;
        cp->birctr++;
        break;
    case DRRIP:
        switch(cp->sets[set].policy){
            case SRRIP:
                repl->RRPV = cp->RRPV_new;
                psel = minima( psel+1, 1023 );
                break;
            case BRRIP:
                if(cp->birctr%32==0) repl->RRPV = cp->RRPV_new-1;
                else             repl->RRPV = cp->RRPV_new;
                psel = maxima( psel-1, -1024 );
                cp->birctr++;
                break;
        }
        update_DRRIP_sets(cp, psel);
    default: break;
}
```

  - After determining the RRPV value, we should update the policy of follower sets depends on modified **psel**

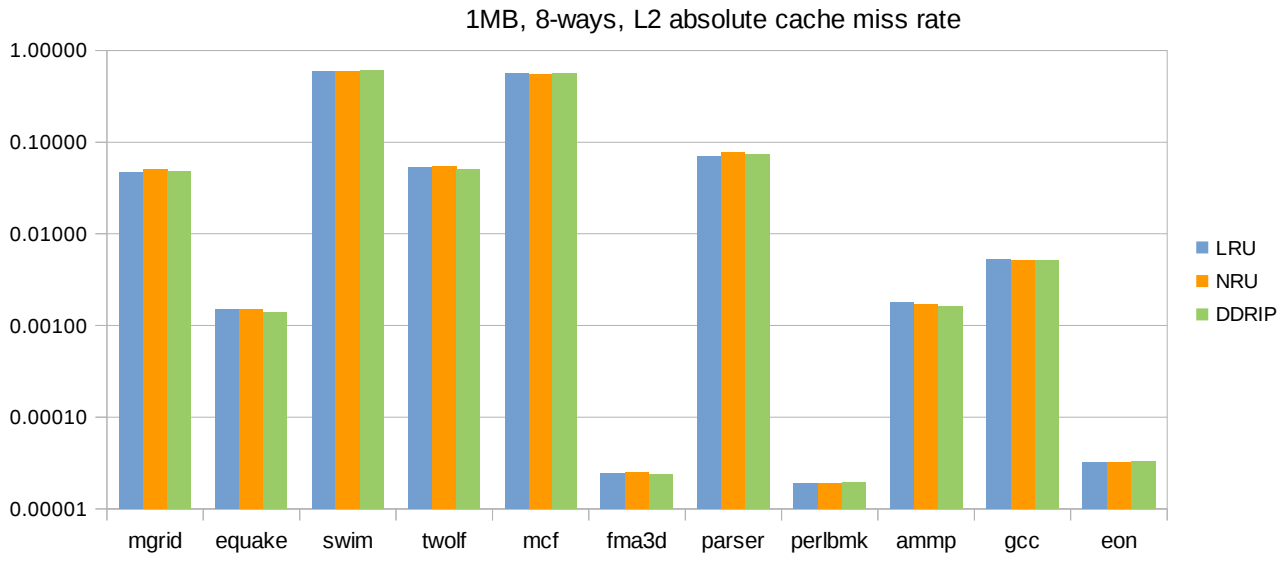
- Set RRPV value of re-reference block on cache hit
  - Same as Not-Recently-Used and BRRIP

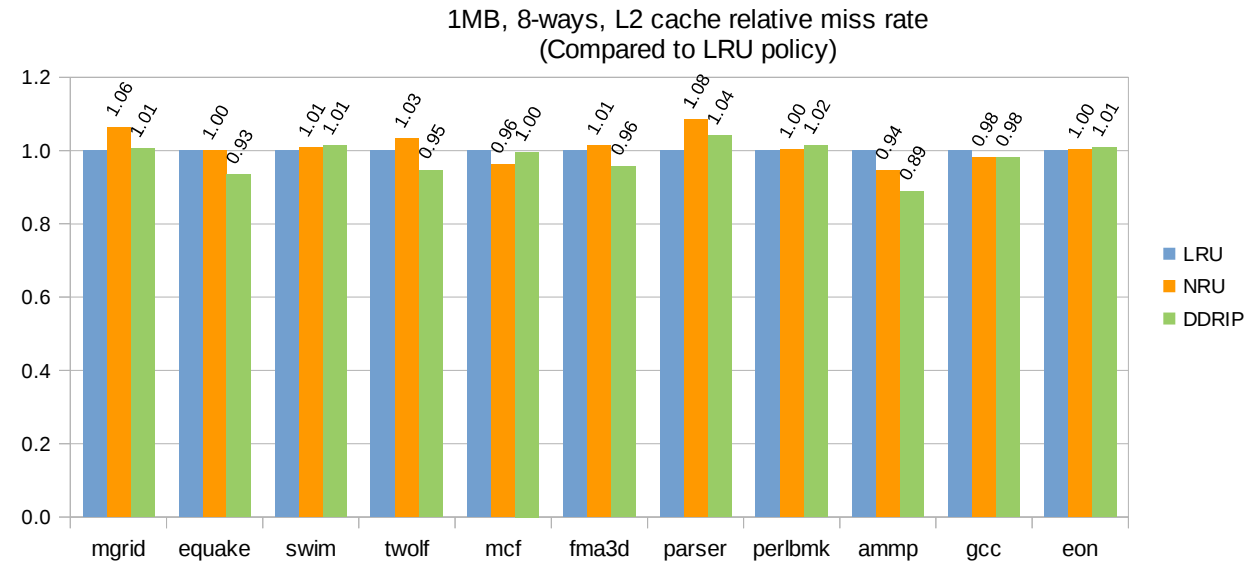
iv. update_DRRIP_sets

- **flr_policy** is the follower sets replacement policy, which is depends on **psel**
- **period** is the period of the dedicated sets, used to calculate which one is the dedicated sets
- Change the replacement policy only if its index is mapped to follower sets, i.e. **i%period!=0** (dedicated BRRIP) and **i%period!=1** (dedicated SRRIP)

```
void
update_DRRIP_sets(struct cache_t* cp, int psel){
    int i;
    int period = cp->nsets/32;
    enum cache_policy flr_policy = (psel>0)?BRRIP:SRRIP;
    for(i=0; i<cp->nsets; i++){
        if(i%period!=0 && i%period!=1) cp->sets[i].policy = flr_policy;
    }
    return;
}
```

3. Result about replacement policy
   a) 1MB, 8-ways, L2 cache miss rate

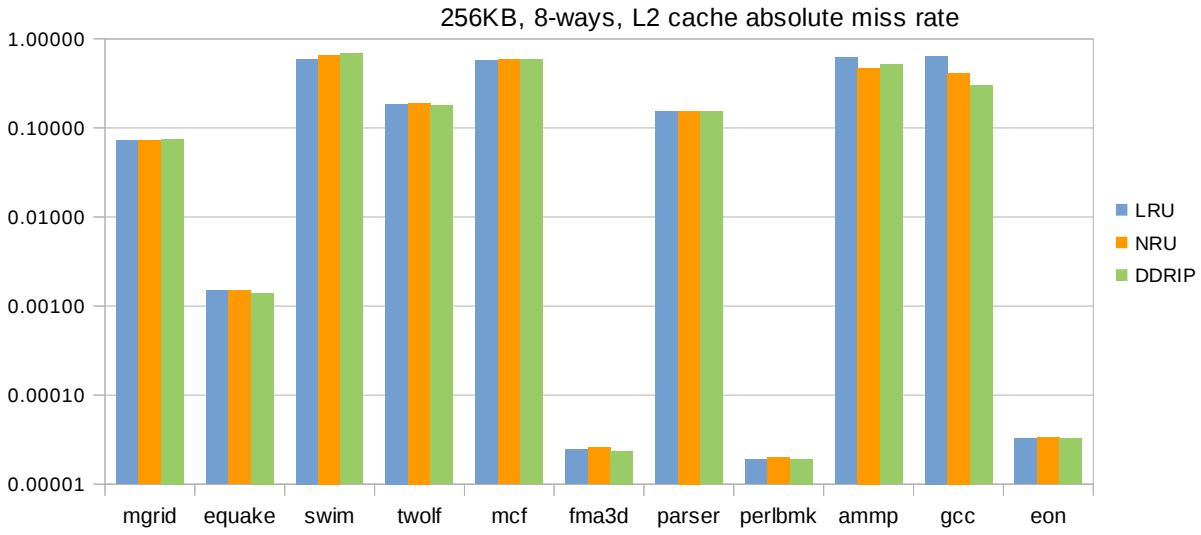


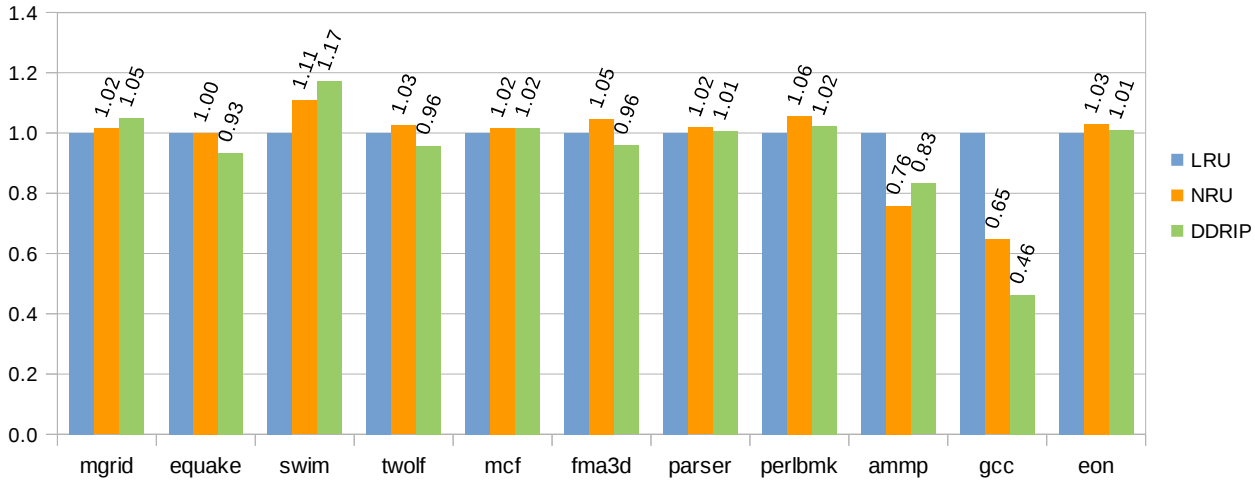


Policy	mgrid	equake	swim	twolf	mcf	fma3d	parser	perlbmk	ammp	gcc	eon
LRU	**1.0000**	1.0000	**1.0000**	1.0000	1.0000	1.0000	**1.0000**	**1.0000**	1.0000	1.0000	**1.0000**
NRU	1.0636	1.0000	1.0080	1.0322	**0.9619**	1.0129	1.0849	1.0038	0.9444	0.9808	1.0036
DDRIP	1.0064	**0.9333**	1.0134	**0.9470**	0.9959	**0.9567**	1.0410	1.0152	**0.8889**	**0.9808**	1.0090

- Compared with LRU, NRU doesn't improve miss rate so much. Even in some cases, LRU has better miss rate than NRU
- Compared with LRU, DRRIP have much better miss rate in some benchmarks. (There are three benchmarks improving more than 5%). Therefore, the highest miss rate in DRRIP is smaller than 1.05. Show that the miss rate in DRRIP is not worse than over 5% in all benchmarks. It means in large cache size, DRRIP may increase miss rate little in some cases, but it can reduce much miss rate in other cases.

b) 256KB, 8-ways, L2 cache miss rate

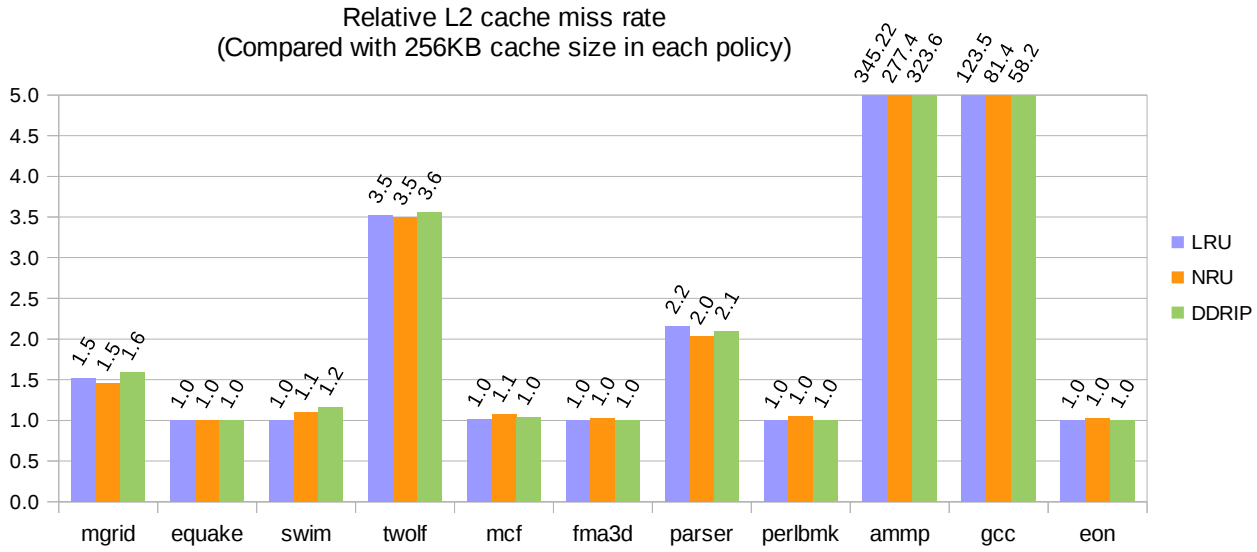




Policy	mgrid	equake	swim	twolf	mcf	fma3d	parser	perlbmk	ammp	gcc	eon
LRU	**1.0000**	1.0000	**1.0000**	1.0000	**1.0000**	1.0000	**1.0000**	**1.0000**	1.0000	1.0000	**1.0000**
NRU	1.0153	1.0000	1.1083	1.0253	1.0160	1.0461	1.0183	1.0564	**0.7588**	0.6459	1.0293
DDRIP	1.0505	**0.9333**	1.1711	**0.9580**	1.0166	**0.9585**	1.0065	1.0229	0.8331	**0.4623**	1.0102

- Compared with LRU, NRU doesn't improve miss rate in most of cases, but have great improvement in two cases.
- Compared with LRU, DRRIP have great improvement (>15%) in ammp and gcc. Therefore, except swim benchmark, the others relative miss rate is smaller or close to 1.05. It means in smaller cache size, DRRIP may increase miss rate little in some cases, but it can strongly reduce miss rate in other cases.

4. Result about cache size



Policy	mgrid	equake	swim	twolf	mcf	fma3d	parser	perlbmk	ammp	gcc	eon
LRU	1.5233	1.0000	1.0002	3.5189	1.0178	1.0000	2.1627	0.9962	345.2222	123.5385	1.0000
NRU	1.4542	1.0000	1.0998	3.4954	1.0750	1.0328	2.0300	1.0484	277.3529	81.3529	1.0256
DDRIP	1.5901	1.0000	1.1559	3.5600	1.0390	1.0018	2.0910	1.0037	323.5625	58.2353	1.0012

- In ammp and gcc, the working set may be larger than 256KB, which greatly affect the miss rate of 256KB. However, LRU policy has bad ability to resist big working set, or in ISCA 2010, they say that NRU and DRRIP are thrashing-resistant

5. Conclusion
   a) Large size cache
      - Miss rate: NRU > LRU > DRRIP
      - DRRIP have some improvement compared with LRU and NRU policy
   b) Small size cache
      - Miss rate
        i. ammp, gcc (thrashing case): LRU > NRU = DRRIP
        ii. The others (non-thrashing case): NRU > LRU = DRRIP
      - NRU and DRRIP are thrashing-resistant