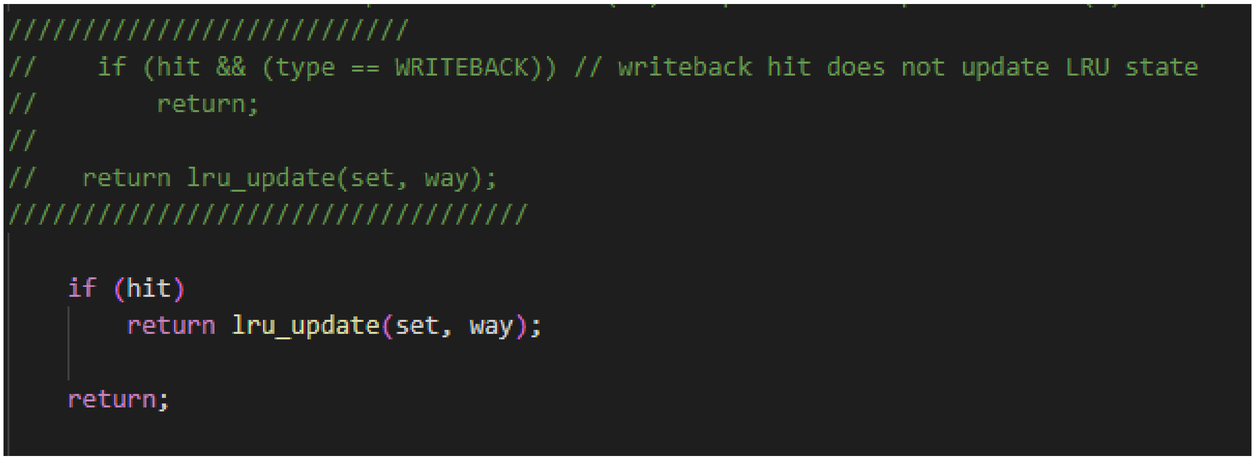
Assignment 2 Report

CPEN411

Sizhe Yan 22164982

1. LIP

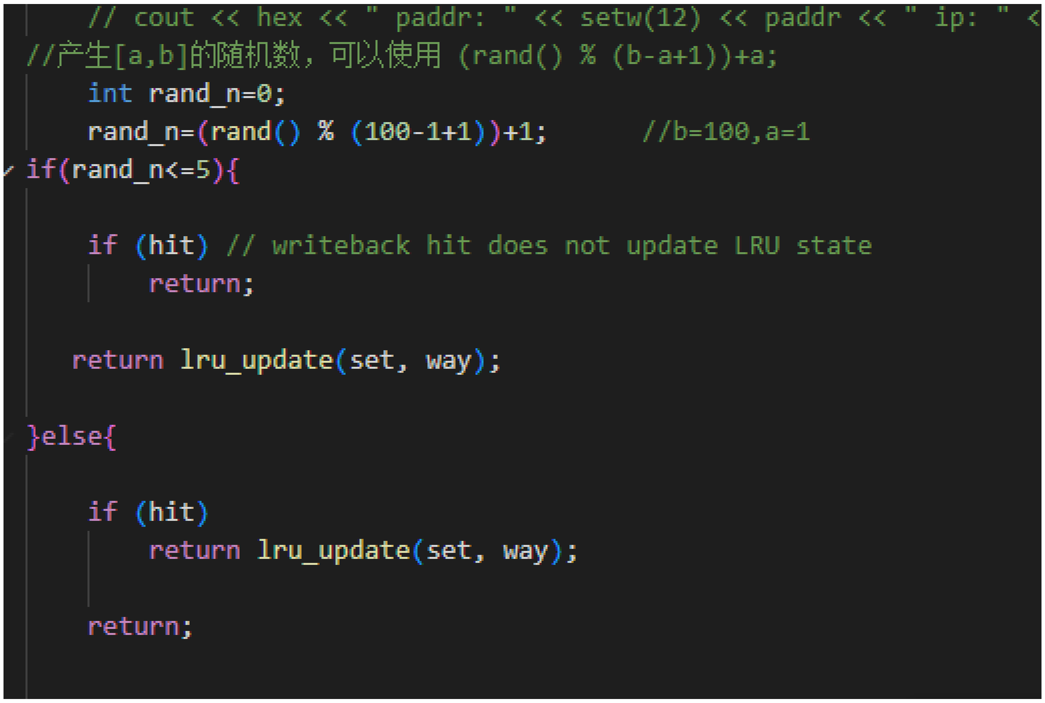
The idea of LIP is push the incoming instruction to MRU when it hits. However the LRU always push the instruction to MRU excepts for hits and write back. As a result, the code for LIP can be simply done by change the “return” value



```
///////////////////////////
//    if (hit && (type == WRITEBACK)) // writeback hit does not update LRU state
//        return;
//
//   return lru_update(set, way);
///////////////////////////////////

    if (hit)
        return lru_update(set, way);

    return;
```

2. BIP

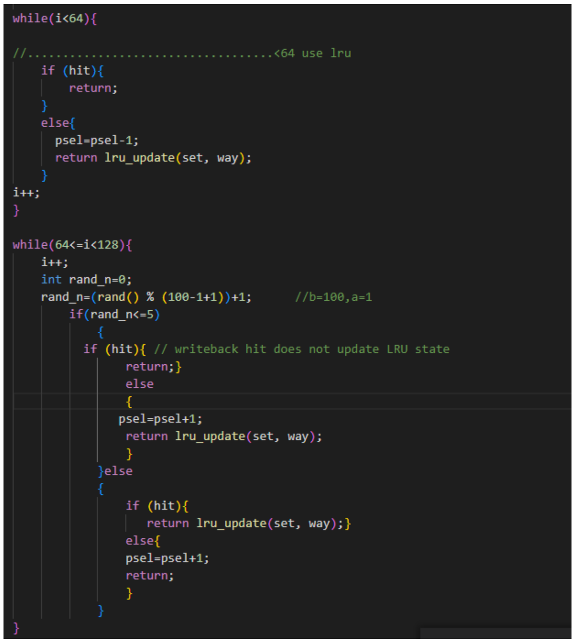
BIP is based on LIP and LRU, as stated in report. From the essay that LIP can be viewed as BIP with epsilon (∈) of 0. LRU can be viewed a s epsilon of 1. As a result, the BIP can be done with a random number generator.



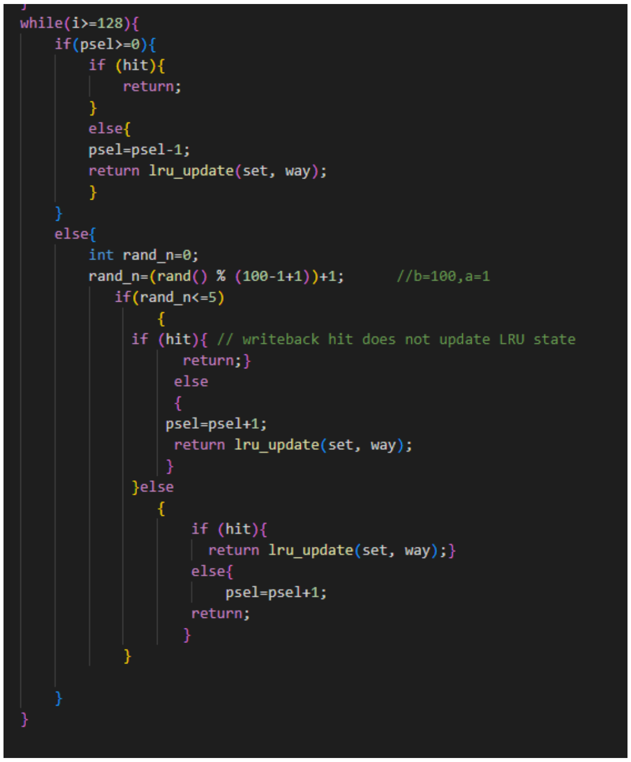
```
    // cout << hex << " paddr: " << setw(12) << paddr << " ip: " <
//产生[a,b]的随机数，可以使用 (rand() % (b-a+1))+a;
    int rand_n=0;
    rand_n=(rand() % (100-1+1))+1;      //b=100,a=1
if(rand_n<=5){

    if (hit) // writeback hit does not update LRU state
        return;

   return lru_update(set, way);

}else{

    if (hit)
        return lru_update(set, way);

    return;
```

3. DIP

The core idea of DIP is find with policy is better, is BIP or LRU. As a result we choose a variable “*psel*” to keep tracking which policy is better. For a miss happened during the execution of LRU, the psel will +1 and a miss happened during the execution of BIP will -1. As a result after a large number of executions, we can check the psel value can see which policy is better. And psel will control the policy that the cpu will use. If psel is >=0, BIP will be executed. If psel <0 then the LRU will be executed



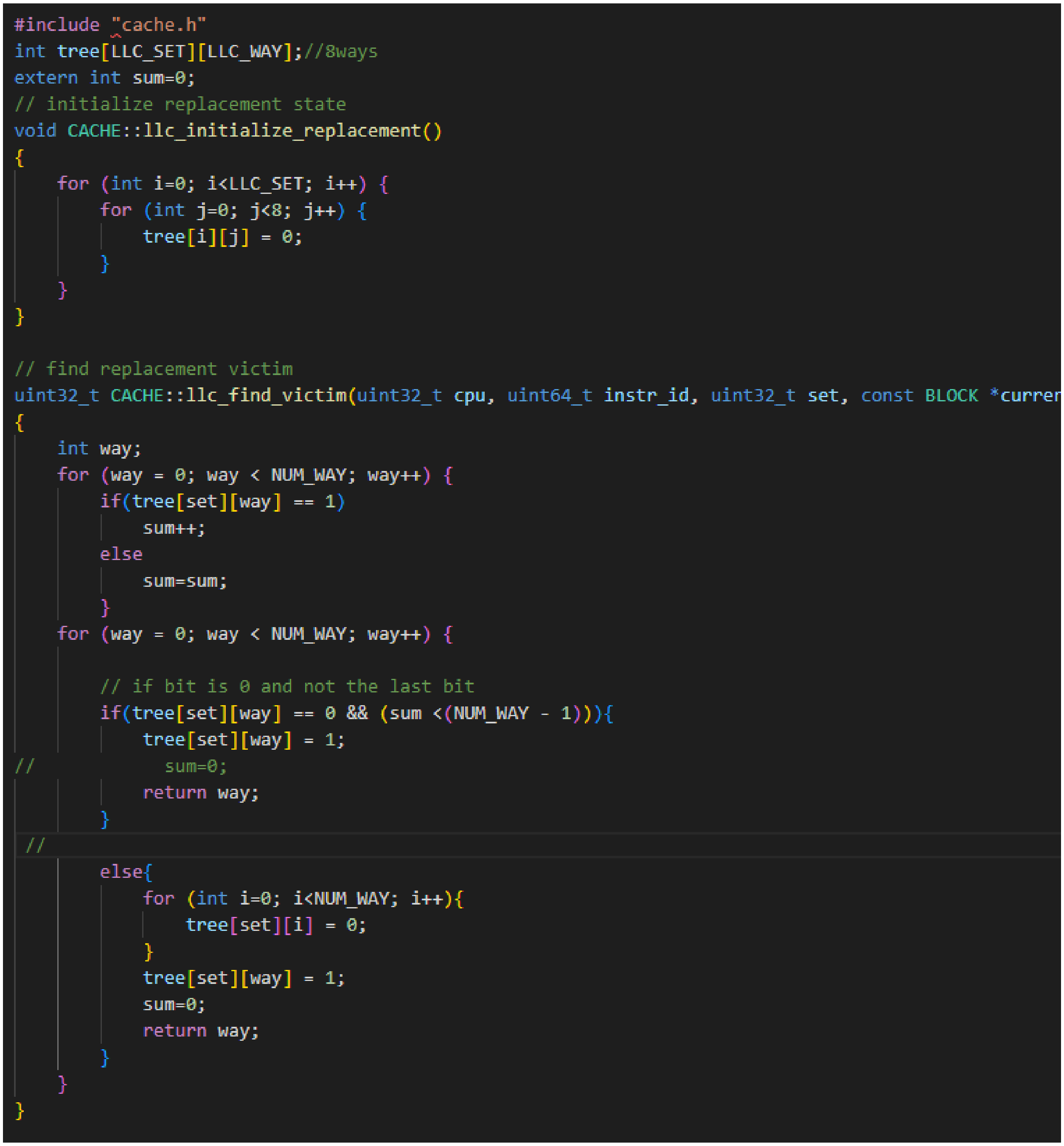
```
while(i<64){

//.................................<64 use lru
    if (hit){
        return;
    }
    else{
      psel=psel-1;
      return lru_update(set, way);
    }
i++;
}

while(64<=i<128){
    i++;
    int rand_n=0;
    rand_n=(rand() % (100-1+1))+1;      //b=100,a=1
        if(rand_n<=5)
            {
          if (hit){ // writeback hit does not update LRU state
                return;}
                else
                {
               psel=psel+1;
                return lru_update(set, way);
                }
            }else
            {
                if (hit){
                   return lru_update(set, way);}
                else{
                psel=psel+1;
                return;
                }
            }
}
```



```
while(i>=128){
    if(psel>=0){
        if (hit){
            return;
        }
        else{
        psel=psel-1;
        return lru_update(set, way);
        }
    }
    else{
        int rand_n=0;
        rand_n=(rand() % (100-1+1))+1;      //b=100,a=1
           if(rand_n<=5)
                {
             if (hit){ // writeback hit does not update LRU state
                   return;}
                  else
                  {
                 psel=psel+1;
                  return lru_update(set, way);
                 }
             }else
                {
                    if (hit){
                      return lru_update(set, way);}
                    else{
                        psel=psel+1;
                    return;
                   }
            }

    }
}
```

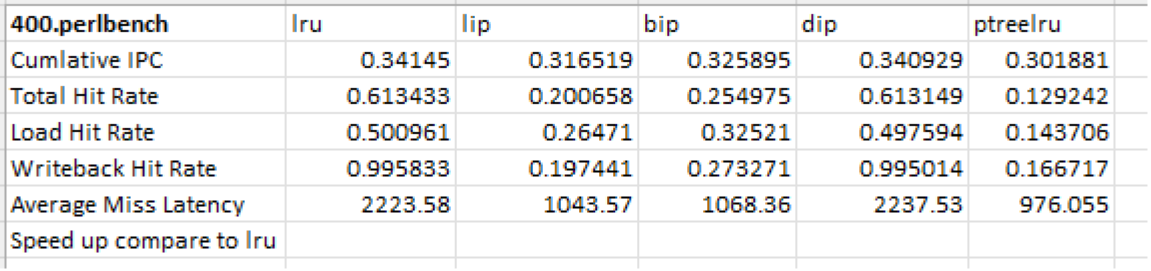
4. pLRU

The idea of pLRU is a tree with the records of of the “cell” is hit. The tag will be 1 after a hit. However when the tree will be full of one after a hit, the tree will be reset to all 0s except the incoming hit. It can be done by using a 2d array.

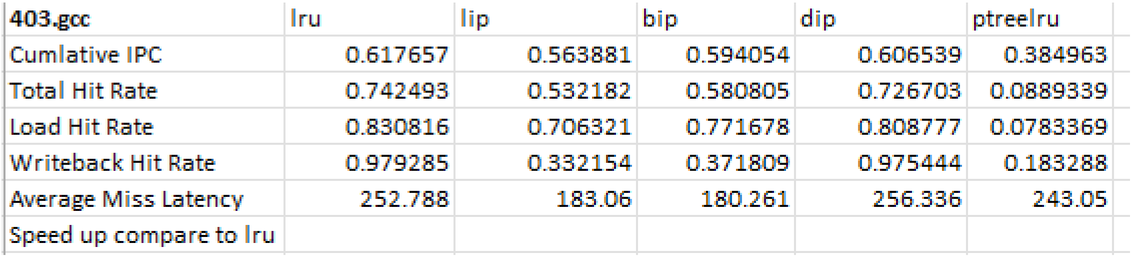


```
#include "cache.h"
int tree[LLC_SET][LLC_WAY];//8ways
extern int sum=0;
// initialize replacement state
void CACHE::llc_initialize_replacement()
{
    for (int i=0; i<LLC_SET; i++) {
        for (int j=0; j<8; j++) {
            tree[i][j] = 0;
        }
    }
}

// find replacement victim
uint32_t CACHE::llc_find_victim(uint32_t cpu, uint64_t instr_id, uint32_t set, const BLOCK *curren
{
    int way;
    for (way = 0; way < NUM_WAY; way++) {
        if(tree[set][way] == 1)
            sum++;
        else
            sum=sum;
        }
    for (way = 0; way < NUM_WAY; way++) {

        // if bit is 0 and not the last bit
        if(tree[set][way] == 0 && (sum <(NUM_WAY - 1))){
            tree[set][way] = 1;
//            sum=0;
            return way;
        }
 //
        else{
            for (int i=0; i<NUM_WAY; i++){
                tree[set][i] = 0;
            }
            tree[set][way] = 1;
            sum=0;
            return way;
        }
    }
}
```

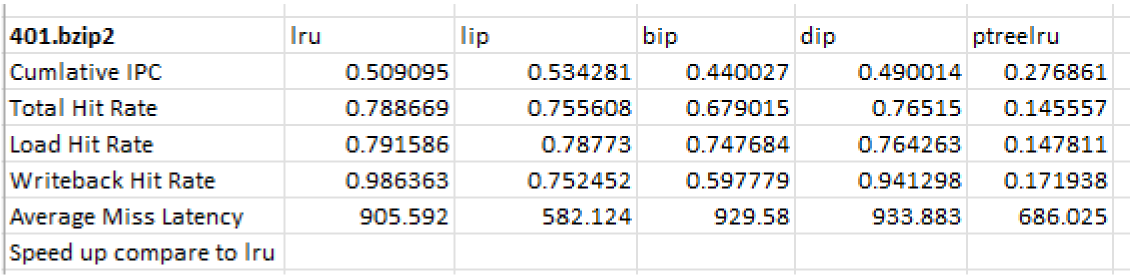
Performance Comparison



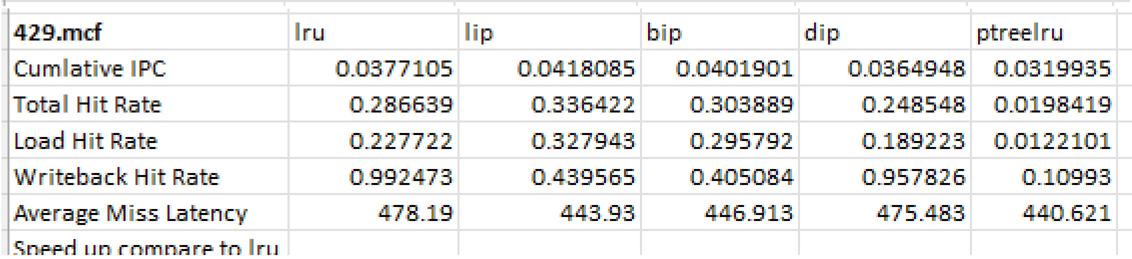
400.perlbench	lru	lip	bip	dip	ptreelru
Cumlative IPC	0.34145	0.316519	0.325895	0.340929	0.301881
Total Hit Rate	0.613433	0.200658	0.254975	0.613149	0.129242
Load Hit Rate	0.500961	0.26471	0.32521	0.497594	0.143706
Writeback Hit Rate	0.995833	0.197441	0.273271	0.995014	0.166717
Average Miss Latency	2223.58	1043.57	1068.36	2237.53	976.055
Speed up compare to lru					



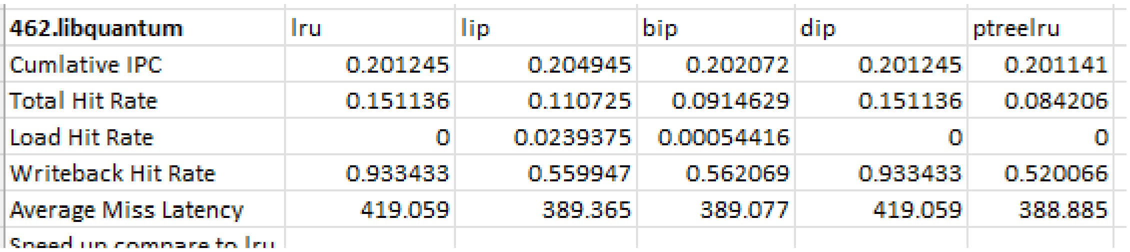
403.gcc	lru	lip	bip	dip	ptreelru
Cumlative IPC	0.617657	0.563881	0.594054	0.606539	0.384963
Total Hit Rate	0.742493	0.532182	0.580805	0.726703	0.0889339
Load Hit Rate	0.830816	0.706321	0.771678	0.808777	0.0783369
Writeback Hit Rate	0.979285	0.332154	0.371809	0.975444	0.183288
Average Miss Latency	252.788	183.06	180.261	256.336	243.05
Speed up compare to lru					



401.bzip2	lru	lip	bip	dip	ptreelru
Cumlative IPC	0.509095	0.534281	0.440027	0.490014	0.276861
Total Hit Rate	0.788669	0.755608	0.679015	0.76515	0.145557
Load Hit Rate	0.791586	0.78773	0.747684	0.764263	0.147811
Writeback Hit Rate	0.986363	0.752452	0.597779	0.941298	0.171938
Average Miss Latency	905.592	582.124	929.58	933.883	686.025
Speed up compare to lru					



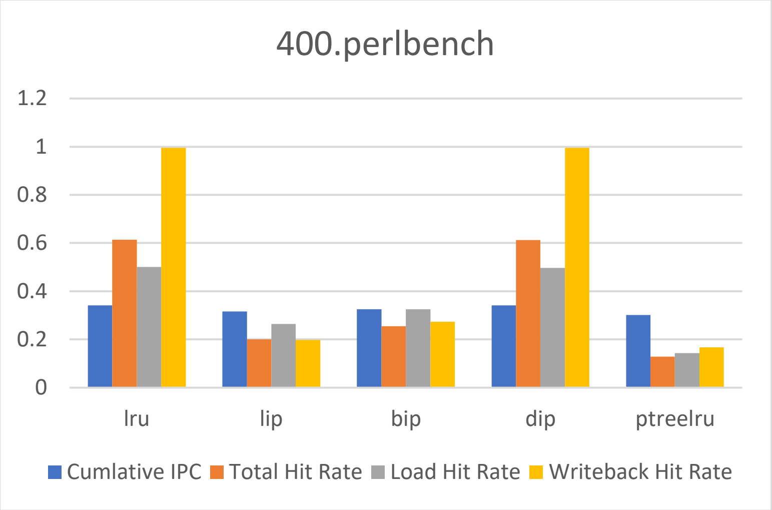
429.mcf	lru	lip	bip	dip	ptreelru
Cumlative IPC	0.0377105	0.0418085	0.0401901	0.0364948	0.0319935
Total Hit Rate	0.286639	0.336422	0.303889	0.248548	0.0198419
Load Hit Rate	0.227722	0.327943	0.295792	0.189223	0.0122101
Writeback Hit Rate	0.992473	0.439565	0.405084	0.957826	0.10993
Average Miss Latency	478.19	443.93	446.913	475.483	440.621
Speed up compare to lru					

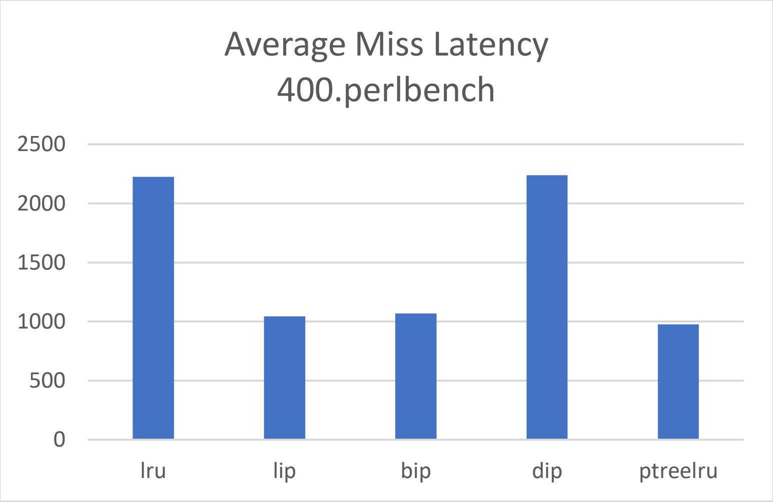


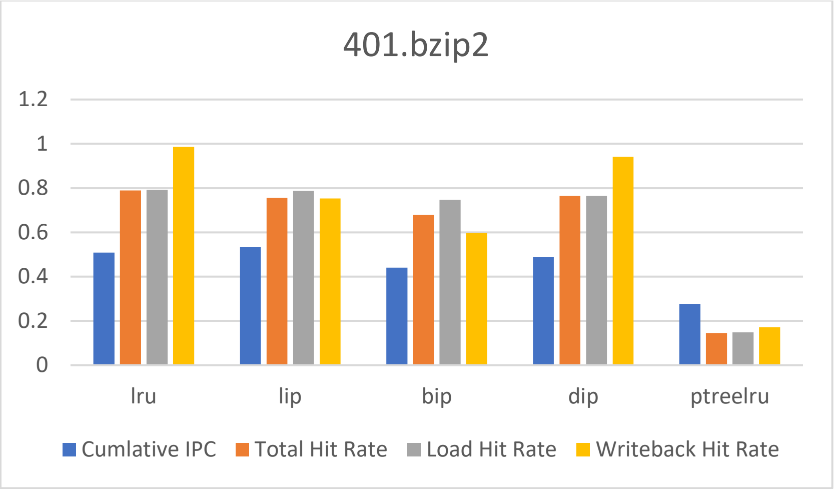
462.libquantum	lru	lip	bip	dip	ptreelru
Cumlative IPC	0.201245	0.204945	0.202072	0.201245	0.201141
Total Hit Rate	0.151136	0.110725	0.0914629	0.151136	0.084206
Load Hit Rate	0	0.0239375	0.00054416	0	0
Writeback Hit Rate	0.933433	0.559947	0.562069	0.933433	0.520066
Average Miss Latency	419.059	389.365	389.077	419.059	388.885
Speed up compare to lru					

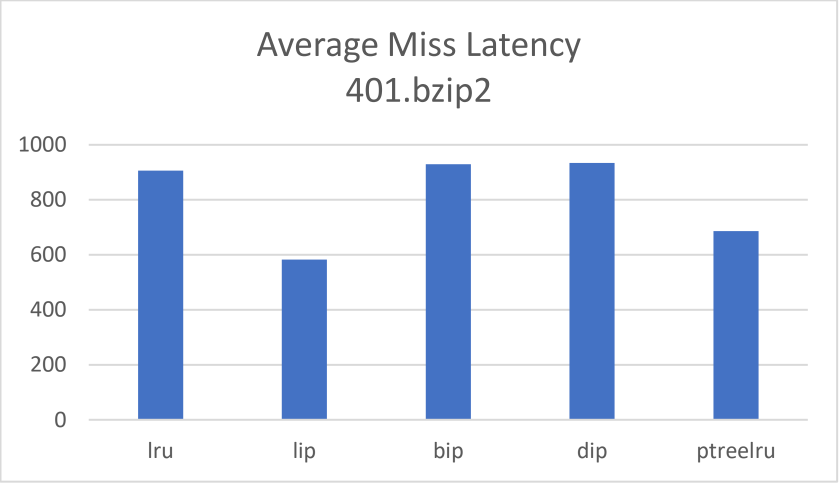
The pictures above shows the data of different tasks for under different policy.

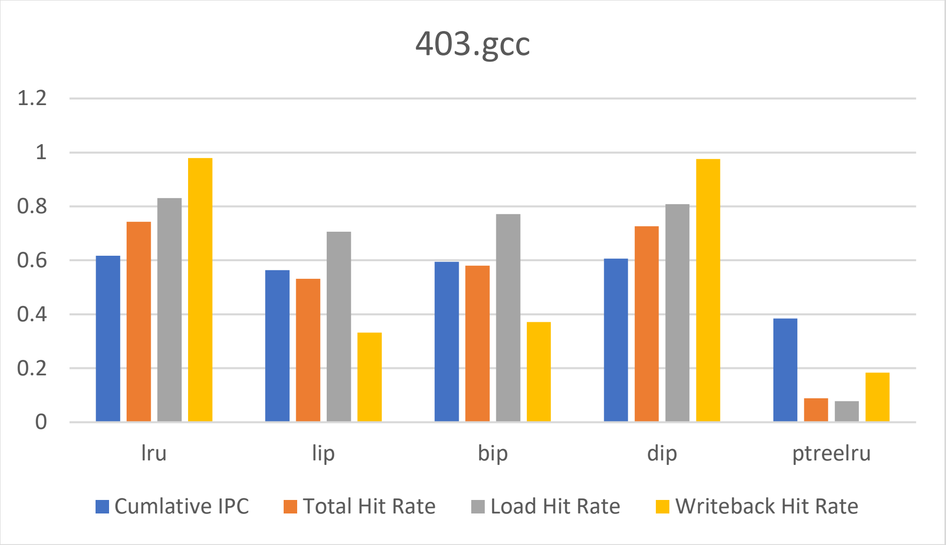
The graphs under indicates the comparison.

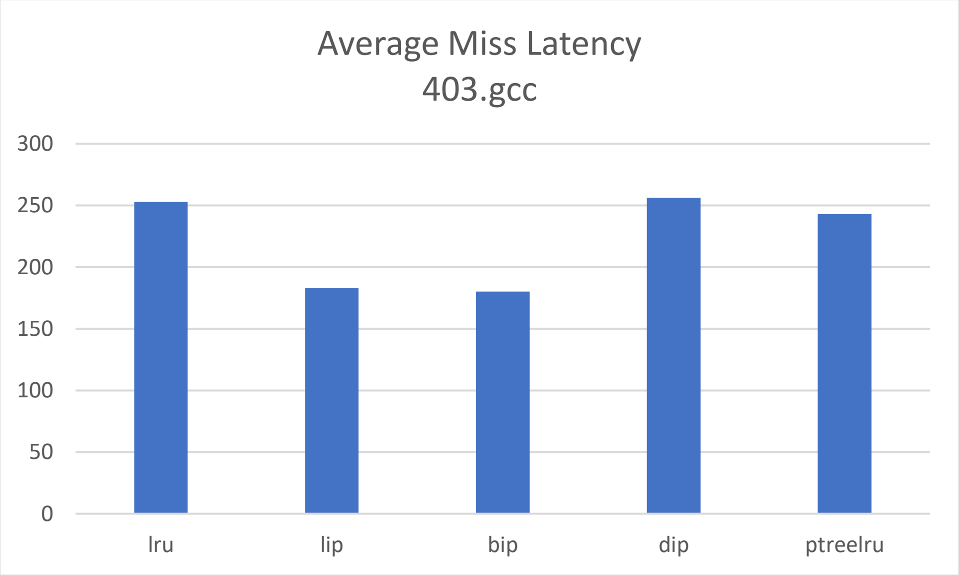

400.perlbench
1.2
1
0.8
0.6
0.4
0.2
0
lru
lip
bip
dip
ptreelru
Cumlative IPC
Total Hit Rate
Load Hit Rate
Writeback Hit Rate

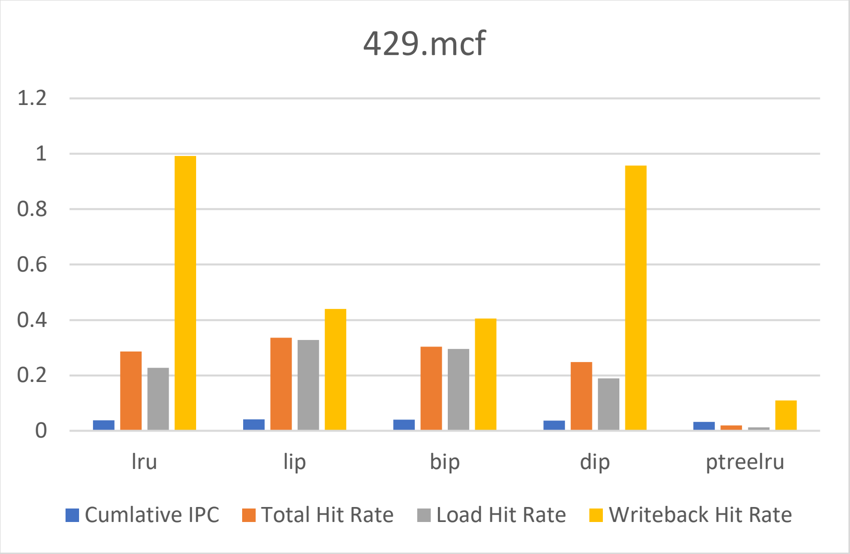

Average Miss Latency
400.perlbench
2500
2000
1500
1000
500
0
lru
lip
bip
dip
ptreelru

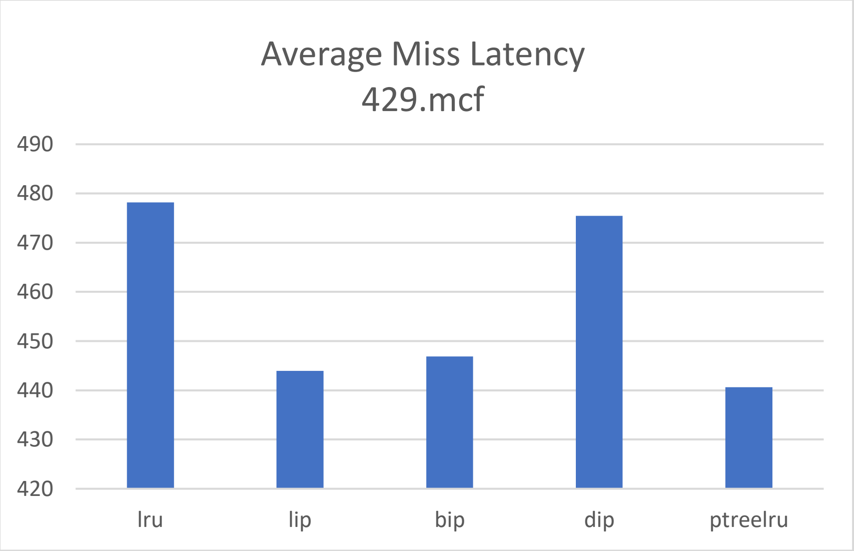

401.bzip2
1.2
1
0.8
0.6
0.4
0.2
0
lru
lip
bip
dip
ptreelru
Cumlative IPC
Total Hit Rate
Load Hit Rate
Writeback Hit Rate

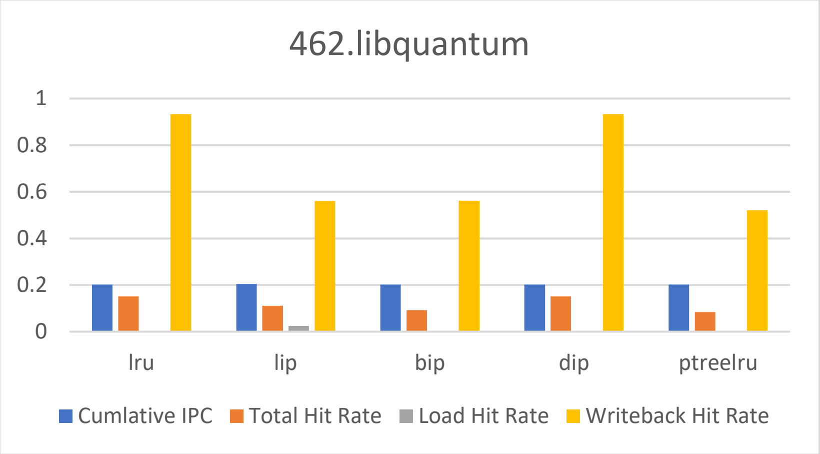

Average Miss Latency
401.bzip2
1000
800
600
400
200
0
lru
lip
bip
dip
ptreelru

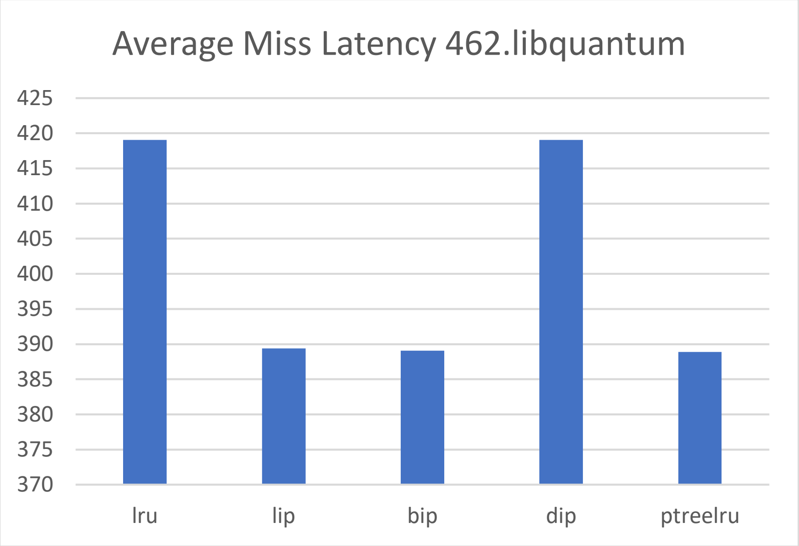

403.gcc
1.2
1
0.8
0.6
0.4
0.2
0
lru
lip
bip
dip
ptreelru
Cumlative IPC
Total Hit Rate
Load Hit Rate
Writeback Hit Rate


Average Miss Latency
403.gcc
300
250
200
150
100
50
0
lru
lip
bip
dip
ptreelru

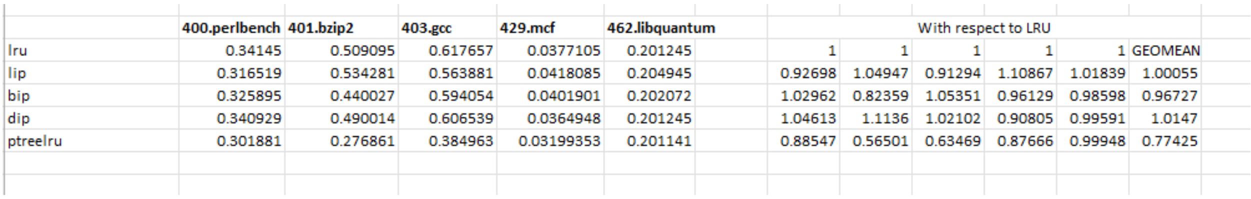

429.mcf
1.2
1
0.8
0.6
0.4
0.2
0
lru
lip
bip
dip
ptreelru
Cumlative IPC
Total Hit Rate
Load Hit Rate
Writeback Hit Rate


Average Miss Latency
429.mcf
490
480
470
460
450
440
430
420
lru
lip
bip
dip
ptreelru





The speed up is indicated by the picture under



	400.perlbench	401.bzip2	403.gcc	429.mcf	462.libquantum		With respect to LRU					
lru	0.34145	0.509095	0.617657	0.0377105	0.201245		1	1	1	1	1	GEOMEAN
lip	0.316519	0.534281	0.563881	0.0418085	0.204945		0.92698	1.04947	0.91294	1.10867	1.01839	1.00055
bip	0.325895	0.440027	0.594054	0.0401901	0.202072		1.02962	0.82359	1.05351	0.96129	0.98598	0.96727
dip	0.340929	0.490014	0.606539	0.0364948	0.201245		1.04613	1.1136	1.02102	0.90805	0.99591	1.0147
ptreelru	0.301881	0.276861	0.384963	0.03199353	0.201141		0.88547	0.56501	0.63469	0.87666	0.99948	0.77425