CPEN411 Assignment4 Report

Sizhe Yan 22164982

2 bit correlated:

```
void O3_CPU::initialize_branch_predictor()
{
    cout << "CPU " << cpu << " Branch predictor" << endl;

    for(int i = 0; i < TABLE_SIZE/2; i++){
        tableNo[cpu][i] = 0;
        tableYes[cpu][i] = 0;
        history[cpu]=0;
    }
}

uint8_t O3_CPU::predict_branch(uint64_t ip)
{
    uint32_t hash = ip % PRIME;
    uint8_t prediction = 0;

    //---- FILL THE PART BELOW THIS ----//

    if((history[cpu])==0){
    if(tableNo[cpu][hash]>1){
        prediction =1;
    }
    else {
        prediction=0;
    }

}
    else{
    if(tableYes[cpu][hash]>1){
        prediction =1;
    }
    else {
        prediction=0;
    }
    }
    return prediction;
}
```

```
if((history[cpu])==0){
    if(taken){
        tableNo[cpu][hash] ++;
    }
    else{
        tableNo[cpu][hash] --;
    }
    if(tableNo[cpu][hash]>MAX_COUNTER){
        tableNo[cpu][hash]=MAX_COUNTER;
    }
    else if(tableNo[cpu][hash]<0){
        tableNo[cpu][hash]=0;
    }
} else{
    if(taken){
        tableYes[cpu][hash] ++;
    }
    else{
        tableYes[cpu][hash] --;
    }
    if(tableYes[cpu][hash]>MAX_COUNTER){
        tableYes[cpu][hash]=MAX_COUNTER;
    }
    else if(tableYes[cpu][hash]<0){
        tableYes[cpu][hash]=0;
    }
}
    (history[cpu])=taken;
    //---- FILL THE PART ABOVE THIS ----//
    return;
```

For part1 2 bit correlated branch predictor, two different table were used in this case, one is for previous prediction is “TAKEN” and the other is for previous prediction is “NOT TAKEN” and they are called tableYes and tableNo respectively. The table size was set to 8191 which won’t excess the size limit for the table. And the 2-bit saturation was considered in decimal: 0 to 3, which means that if the value is less or equal to 1 is not taken, vice versa. For every prediction made, the value will be updated.

hashed_gselect

```
void O3_CPU::initialize_branch_predictor()
{
    cout << "CPU " << cpu << "Branch predictor" << endl;
    for( int k=0;k<NUM_CPUS;k++){
        history[k]=0;
    }

    for(int i = 0; i < TABLE_SIZE; i++)
        table[cpu][i] = 0;



}

uint8_t O3_CPU::predict_branch(uint64_t ip)
{
    uint32_t hash = ip % PRIME;
    uint8_t prediction = 0;

    index_h=history[cpu]^(hash%32);

    //---- FILL THE PART BELOW THIS ----//
    if(table[cpu][index_h*512+hash]>1){
        prediction =1;
    }
    else {
        prediction=0;
    }


    //---- FILL THE PART ABOVE THIS ----//

    return prediction;
```

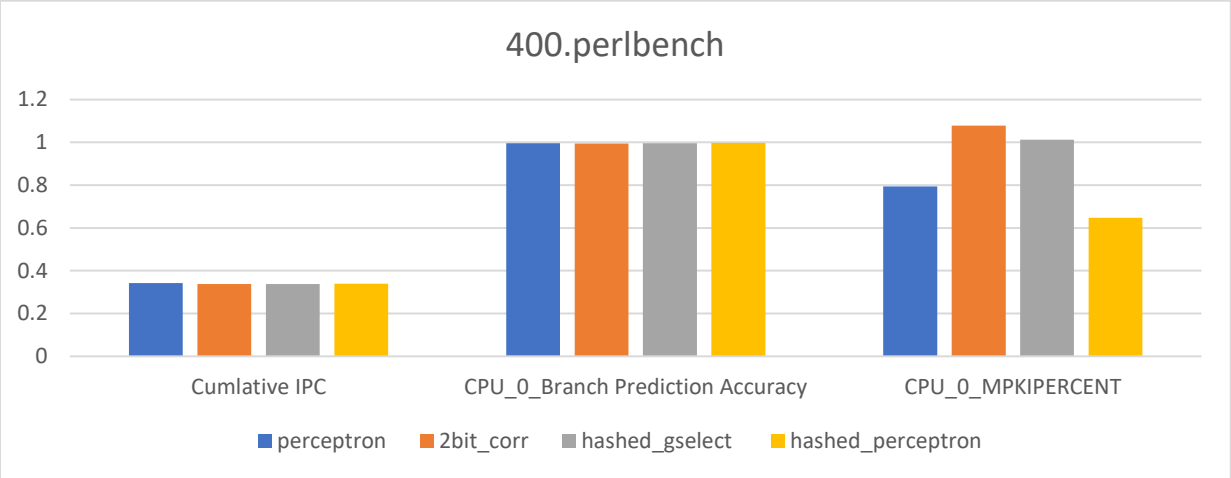
```
void O3_CPU::last_branch_result(uint64_t ip, uint8_t taken)
{
    uint32_t hash = ip % PRIME;

    //---- FILL THE PART BELOW THIS ----//
    if(taken){
        table[cpu][index_h*512+hash] ++;
    }
    else{
        table[cpu][index_h*512+hash] --;
    }
    if(table[cpu][index_h*512+hash]>MAX_COUNTER){
        table[cpu][index_h*512+hash]=MAX_COUNTER;
    }
    else if(table[cpu][index_h*512+hash]<0){
        table[cpu][index_h*512+hash]=0;
    }

    history[cpu]= (((history[cpu] >> 1) + taken * (int)pow(2,4)))%(int)pow(2,5);

    //---- FILL THE PART ABOVE THIS ----//
    return;
}
```

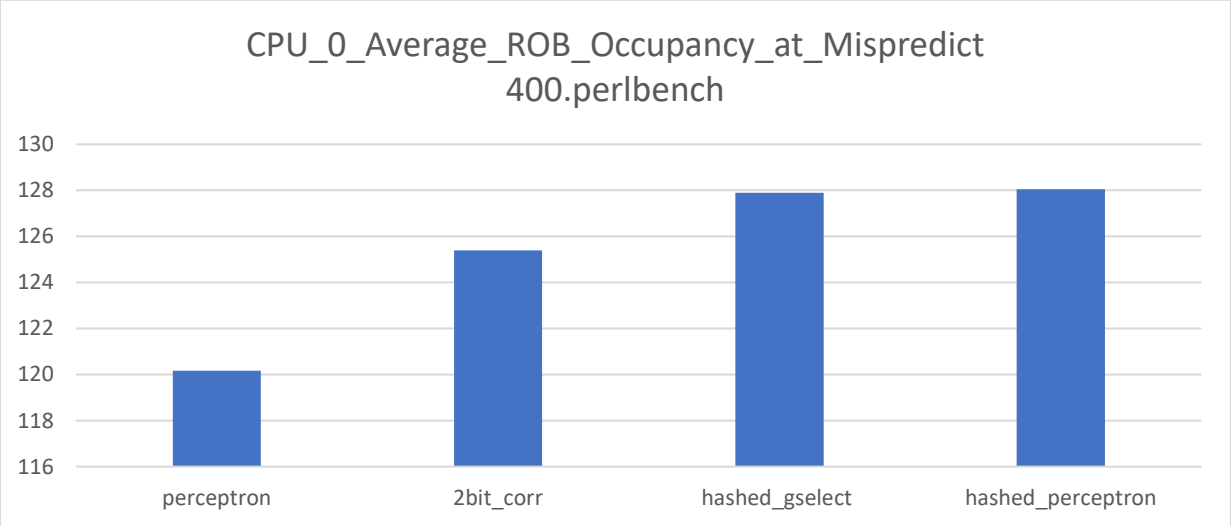
In this question, we are using 5 bits history, which means that we need to have 2^5 tables which is 32. As a result, the 16384 table size is divided into 32 equal tables virtually, and the index to determine which table to update is controlled by the result of “history[cpu]^(hash%32)”, the shift operation is completed by adding “taken*2^4”, the purpose of modulation of 2^5 is we only care about the least 5 bits of the XOR result as the least significant bits have the highest entropy. The update logic is the same as part 1.

Comparison

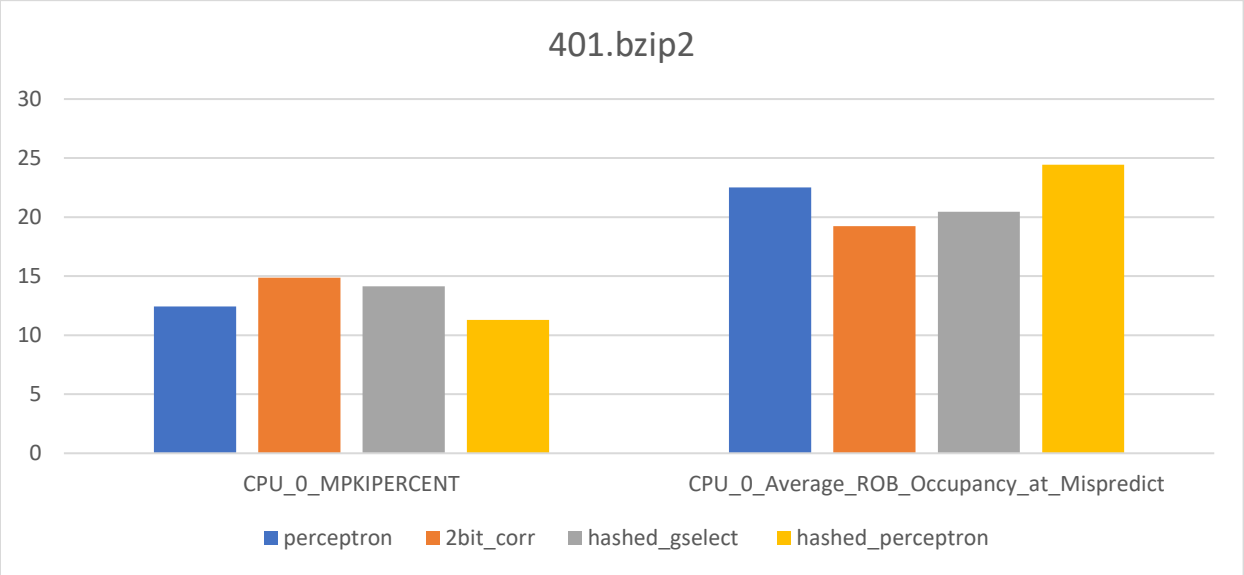
Here is the comparison between different algorithms.

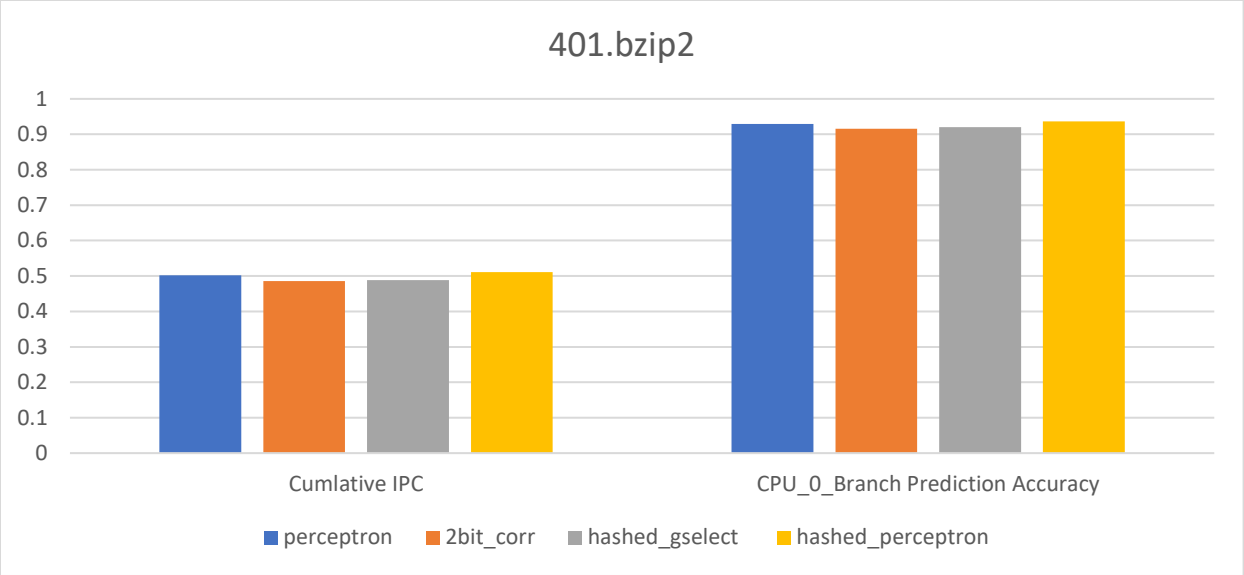
400.perlbench



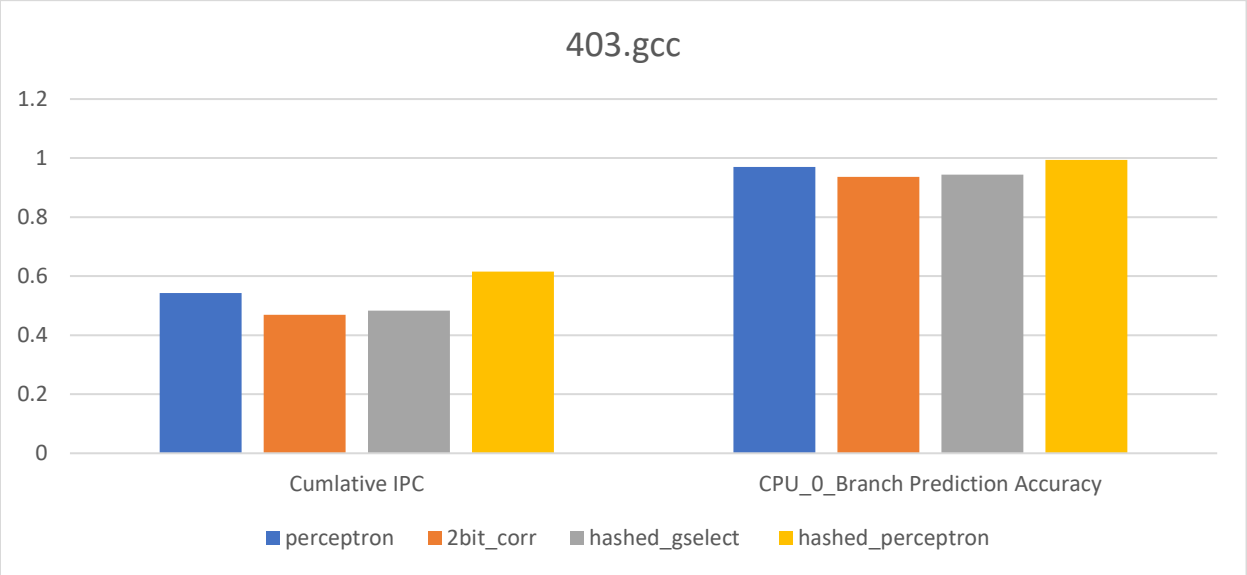


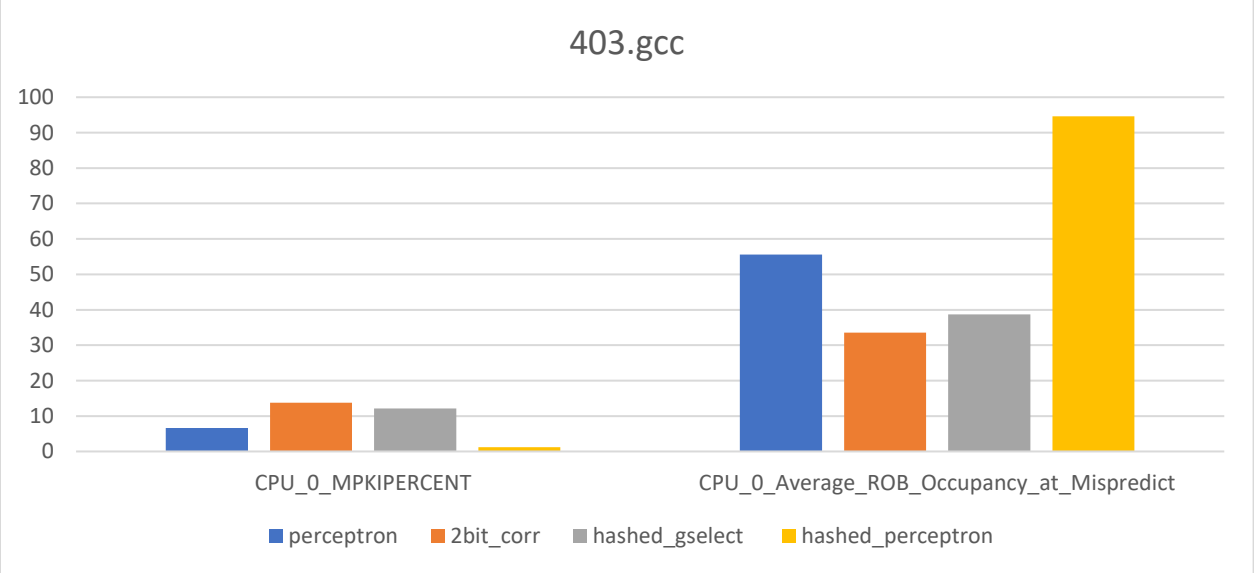
401.bzip2


401.bzip2
30
25
20
15
10
5
0
CPU_0_MPKIPERCENT
CPU_0_Average_ROB_Occupancy_at_Mispredict
perceptron
2bit_corr
hashed_gselect
hashed_perceptron

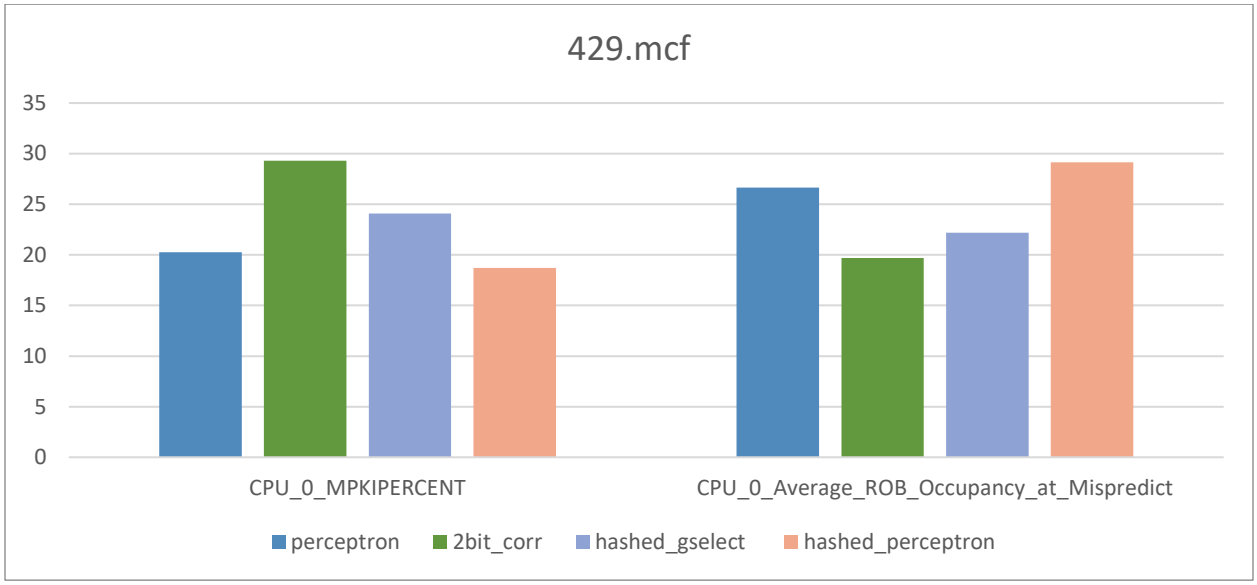

401.bzip2
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
Cumlative IPC
CPU_0_Branch Prediction Accuracy
perceptron
2bit_corr
hashed_gselect
hashed_perceptron

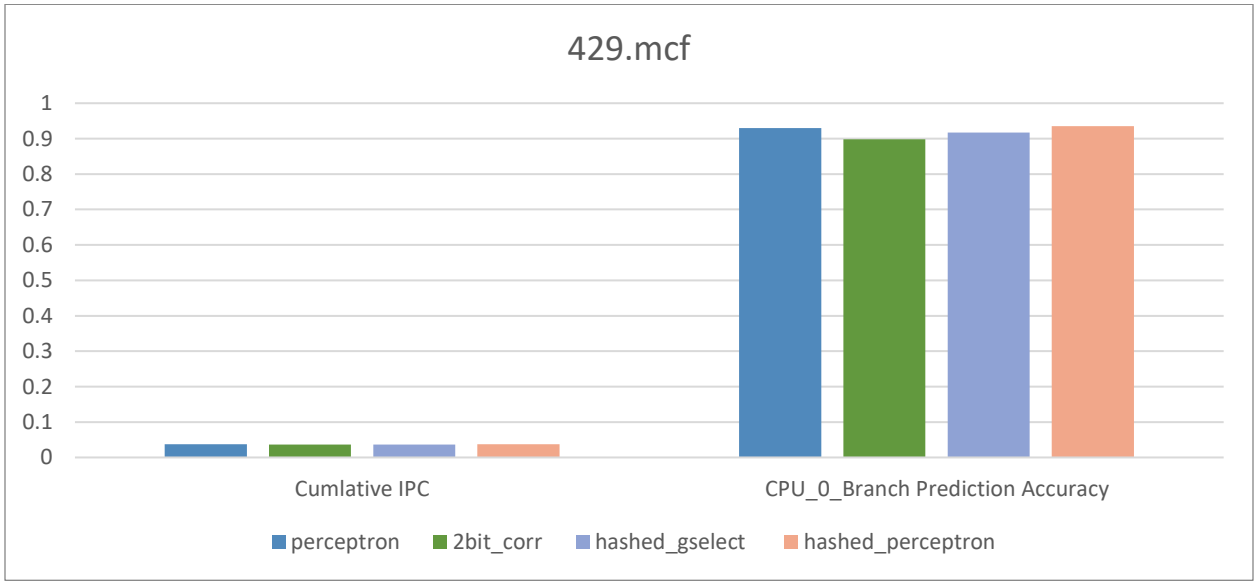
403.gcc


403.gcc
1.2
1
0.8
0.6
0.4
0.2
0
Cumlative IPC
CPU_0_Branch Prediction Accuracy
perceptron
2bit_corr
hashed_gselect
hashed_perceptron

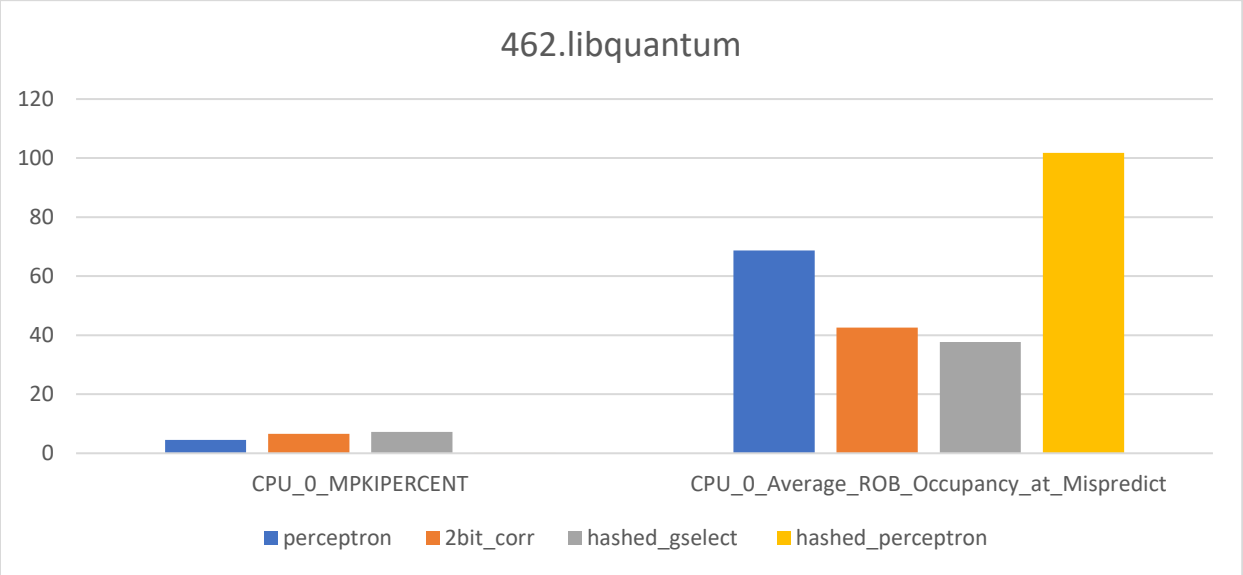

403.gcc
100
90
80
70
60
50
40
30
20
10
0
CPU_0_MPKIPERCENT
CPU_0_Average_ROB_Occupancy_at_Mispredict
perceptron
2bit_corr
hashed_gselect
hashed_perceptron

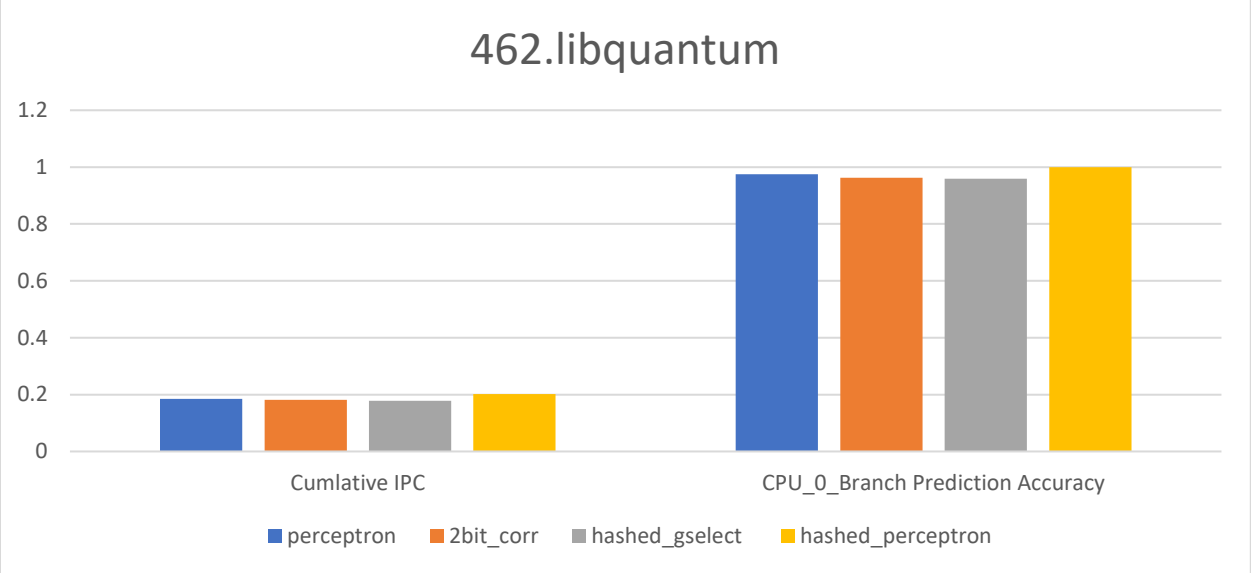
429.mcf


429.mcf
35
30
25
20
15
10
5
0
CPU_0_MPKIPERCENT
CPU_0_Average_ROB_Occupancy_at_Mispredict
perceptron
2bit_corr
hashed_gselect
hashed_perceptron


429.mcf
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
Cumlative IPC
CPU_0_Branch Prediction Accuracy
perceptron
2bit_corr
hashed_gselect
hashed_perceptron

462.libquantum





Speed Up

	400.perlbench	401.bzip2	403.gcc	429.mcf	462.libquantum
perceptron	0.34261	0.501592	0.542396	0.0373335	0.18534
2bit_corr	0.338499	0.485529	0.468824	0.0363028	0.18128
hashed_gselect	0.337808	0.488106	0.483138	0.0367602	0.178577
hashed_perceptron	0.339681	0.51132	0.615946	0.037663	0.201646

						GEOMEAN
With respect to LRU						
	1.00862	0.98097	0.88059	0.99125	0.91914	0.95487
	0.99652	0.94956	0.76114	0.96388	0.899	0.91002
	0.99449	0.9546	0.78438	0.97603	0.8856	0.91565
	1	1	1	1	1	1