nawaabFetch

Akshat Singh 210020013, Yash Ruhatiya 210050169, Yashwanth Reddy Challa 210050171

IPCP specialized to GRAPH applications

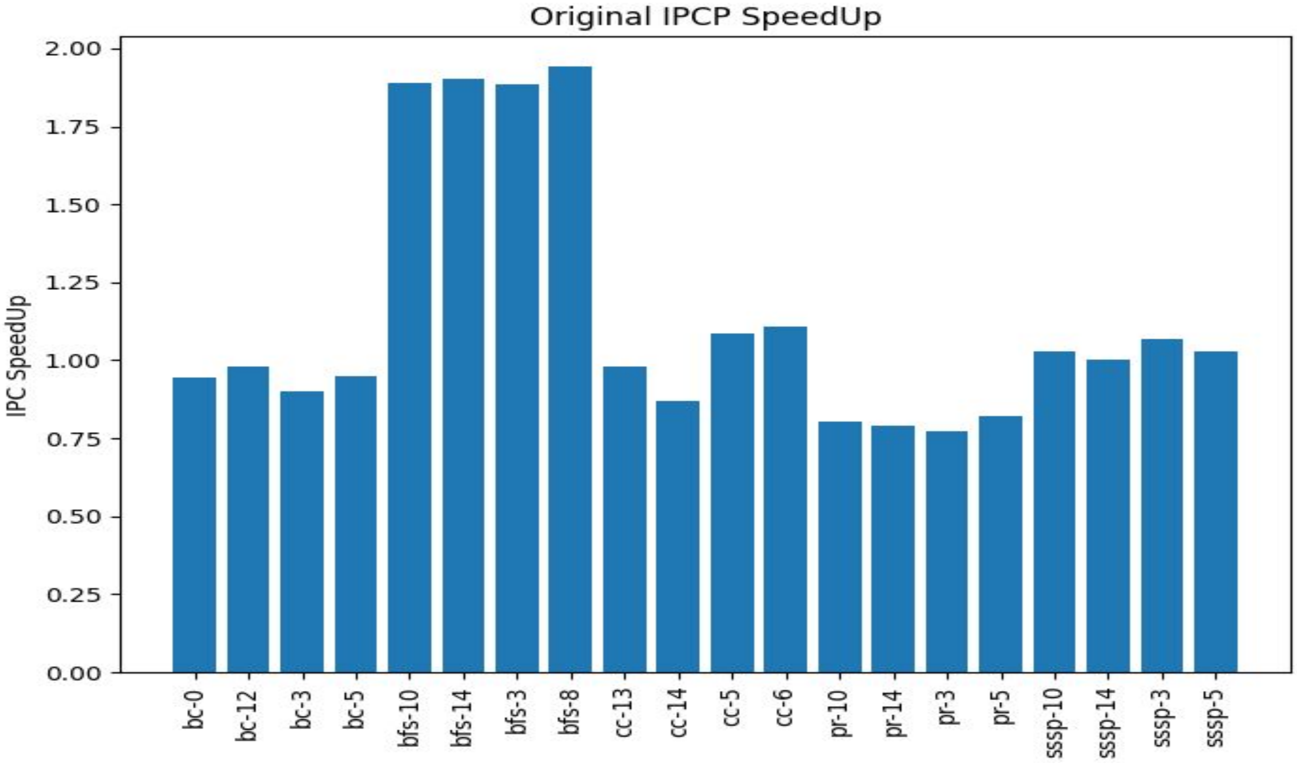
IPCP: Instruction Pointer Classifier based Hardware Prefetching

- **Classify IPs** at **L1D**

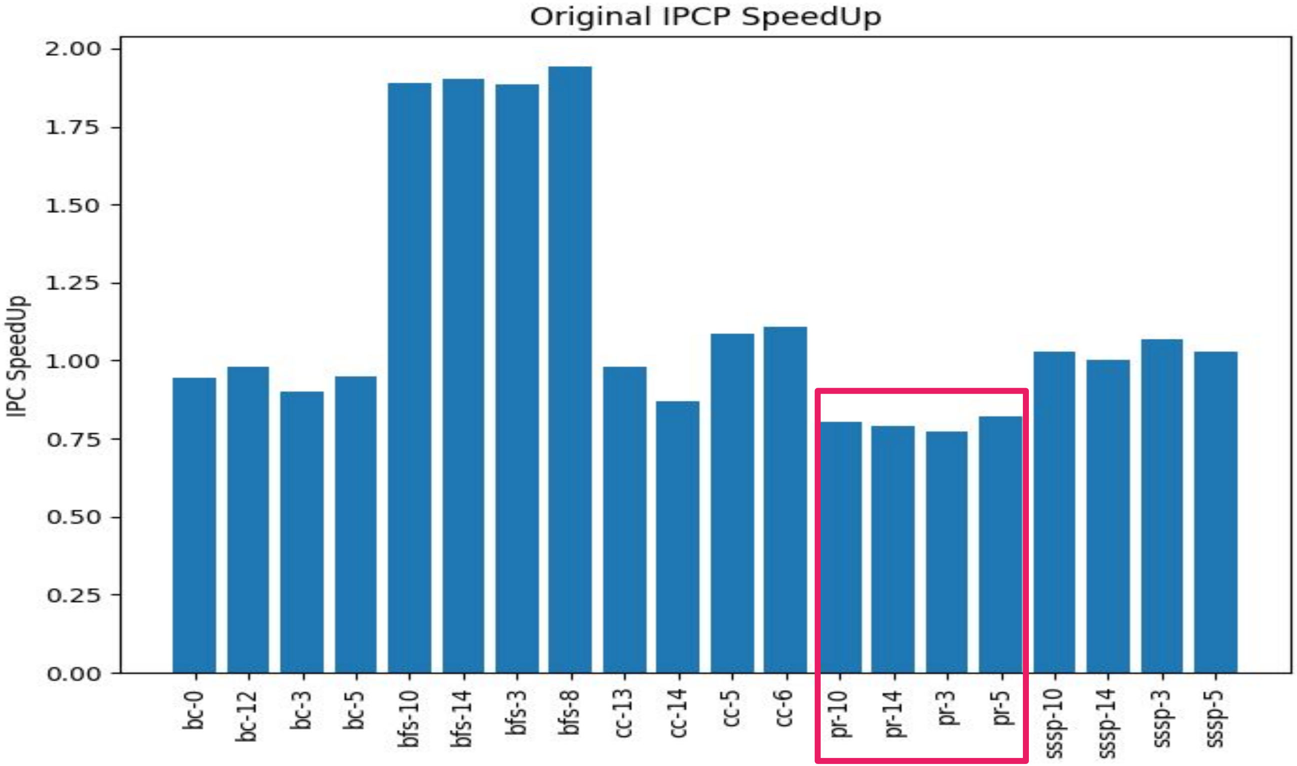
- **Distorted L2C access pattern:** L2C uses **metadata** from L1D

- Last resort: **NL** prefetching

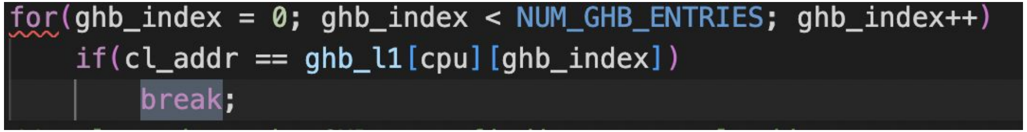
Analyzing the graph traces:



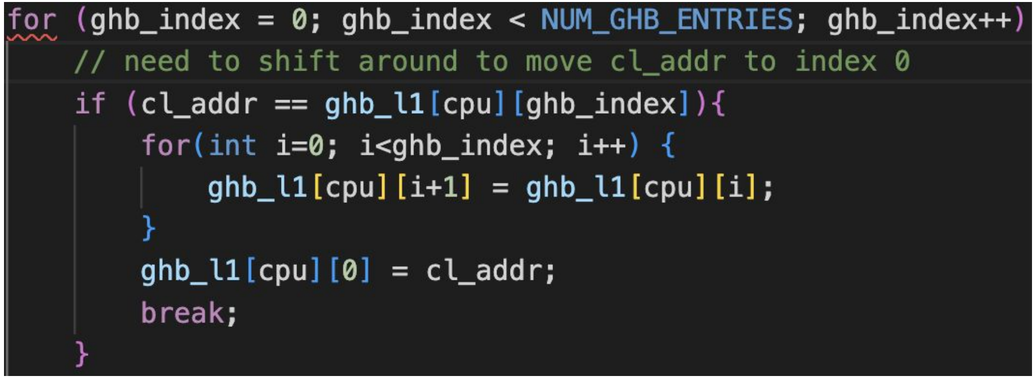
Analyzing the graph traces:



1) NEUTRAL CHANGE: Improvement to GHB



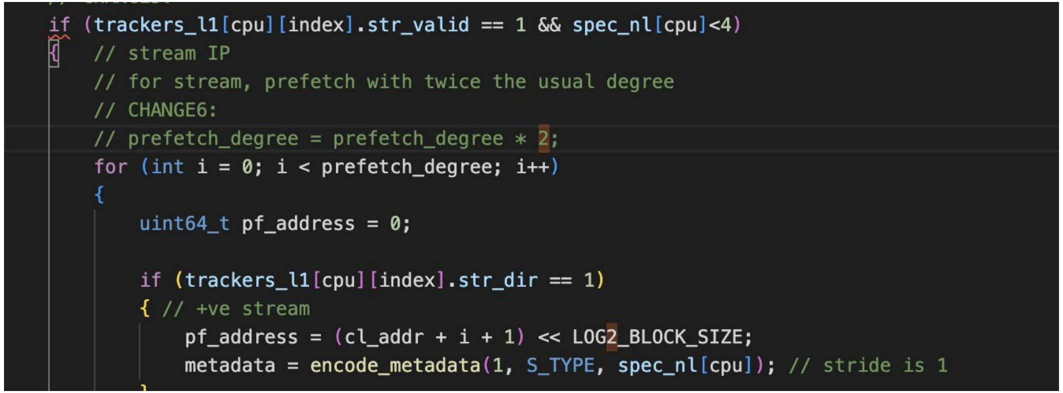
```
for(ghb_index = 0; ghb_index < NUM_GHB_ENTRIES; ghb_index++)
    if(cl_addr == ghb_l1[cpu][ghb_index])
        break;
```



```
for (ghb_index = 0; ghb_index < NUM_GHB_ENTRIES; ghb_index++)
    // need to shift around to move cl_addr to index 0
    if (cl_addr == ghb_l1[cpu][ghb_index]){
        for(int i=0; i<ghb_index; i++) {
            ghb_l1[cpu][i+1] = ghb_l1[cpu][i];
        }
        ghb_l1[cpu][0] = cl_addr;
        break;
    }
```

- Old GHB **did not** maintain perfect order
- New GHB stores the **most recent accesses** at the lowest index
- **No noticeable improvement** in IPC

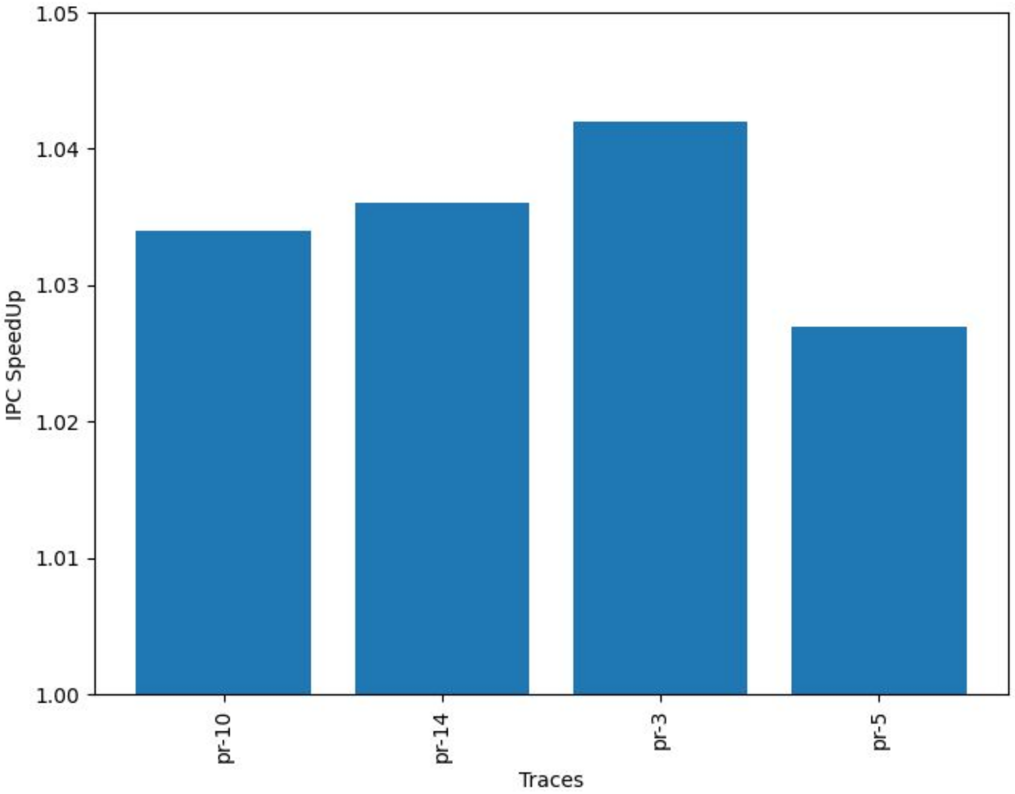
2) POSITIVE CHANGE: Reducing GS Prefetch Degree



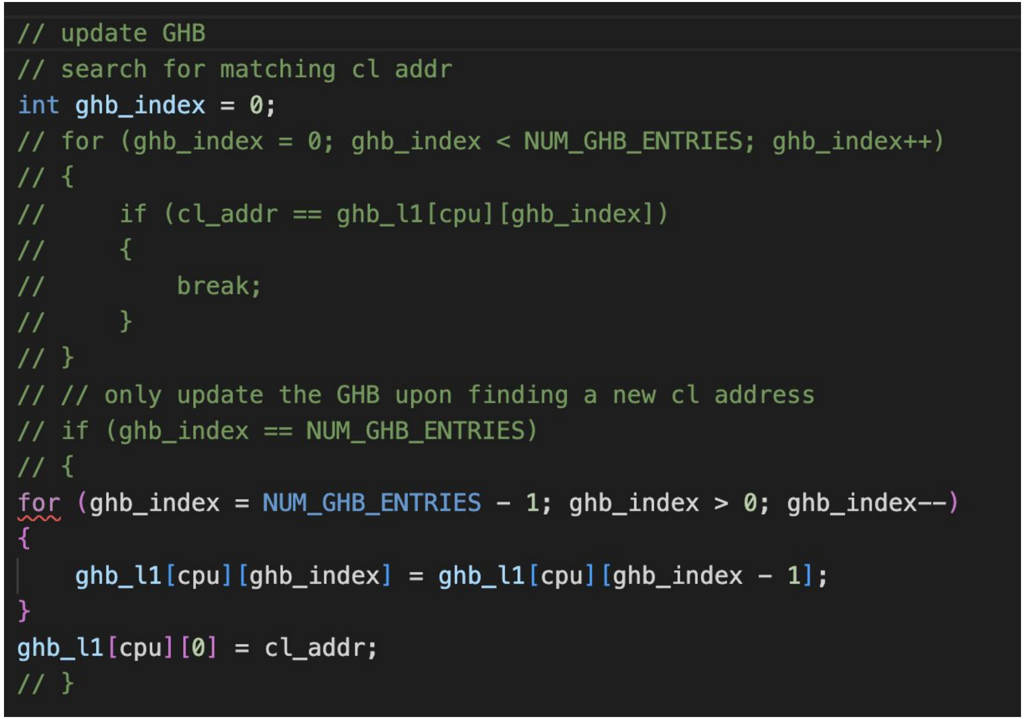
```
if (trackers_l1[cpu][index].str_valid == 1 && spec_nl[cpu]<4)
{   // stream IP
    // for stream, prefetch with twice the usual degree
    // CHANGE6:
    // prefetch_degree = prefetch_degree * 2;
    for (int i = 0; i < prefetch_degree; i++)
    {
        uint64_t pf_address = 0;

        if (trackers_l1[cpu][index].str_dir == 1)
        { // +ve stream
            pf_address = (cl_addr + i + 1) << LOG2_BLOCK_SIZE;
            metadata = encode_metadata(1, S_TYPE, spec_nl[cpu]); // stride is 1
        }
```

- **Don’t multiply** GS prefetch degree by a factor of 2
- L1D **prefetch accuracy was low.** So we tried decreasing the prefetch degrees
- Halving the **GS prefetch degree** did the trick

2) POSITIVE CHANGE: Reducing GS Prefetch Degree



3) POSITIVE CHANGE: Allowing repetition in the GHB

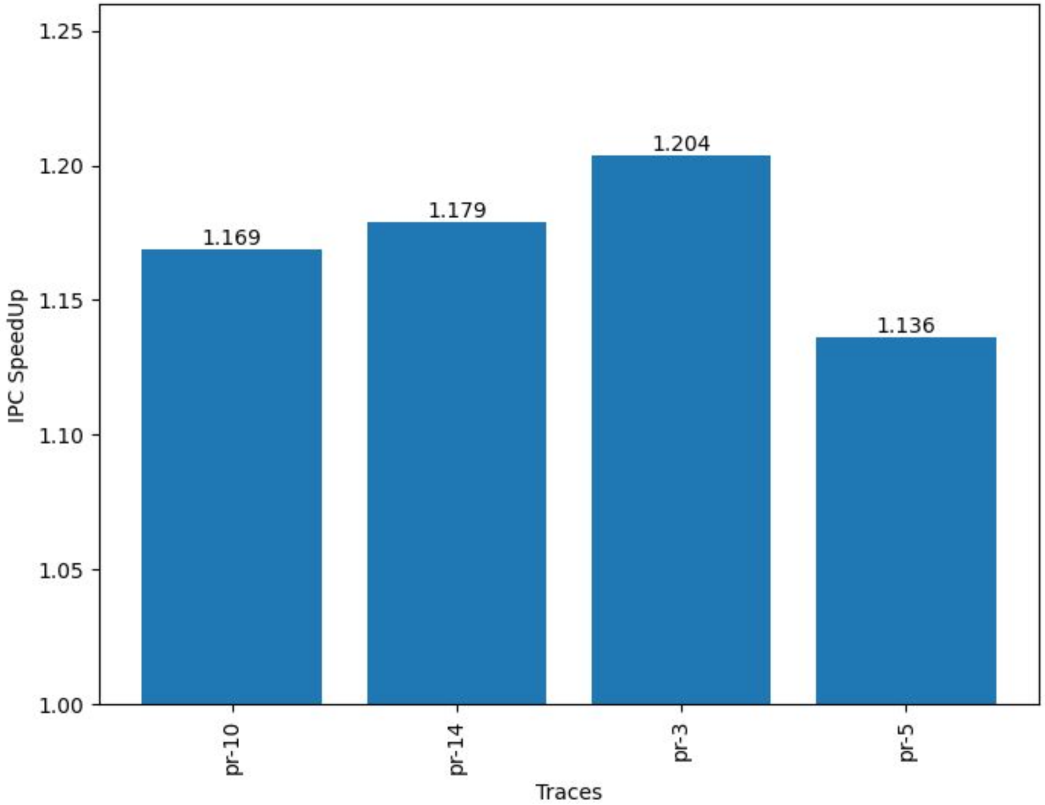


```
// update GHB
// search for matching cl addr
int ghb_index = 0;
// for (ghb_index = 0; ghb_index < NUM_GHB_ENTRIES; ghb_index++)
// {
//     if (cl_addr == ghb_l1[cpu][ghb_index])
//     {
//         break;
//     }
// }
// // only update the GHB upon finding a new cl address
// if (ghb_index == NUM_GHB_ENTRIES)
// {
for (ghb_index = NUM_GHB_ENTRIES - 1; ghb_index > 0; ghb_index--)
{
    ghb_l1[cpu][ghb_index] = ghb_l1[cpu][ghb_index - 1];
}
ghb_l1[cpu][0] = cl_addr;
// }
```

- GHB now simply stores the previous 16 memory accesses, **unique or otherwise**

- **Reduces the likelihood** of an IP being **classified as GS** by reducing the number of unique GHB entries

3) POSITIVE CHANGE: Allowing repetition in the GHB



4) POSITIVE CHANGE: lowering GS priority

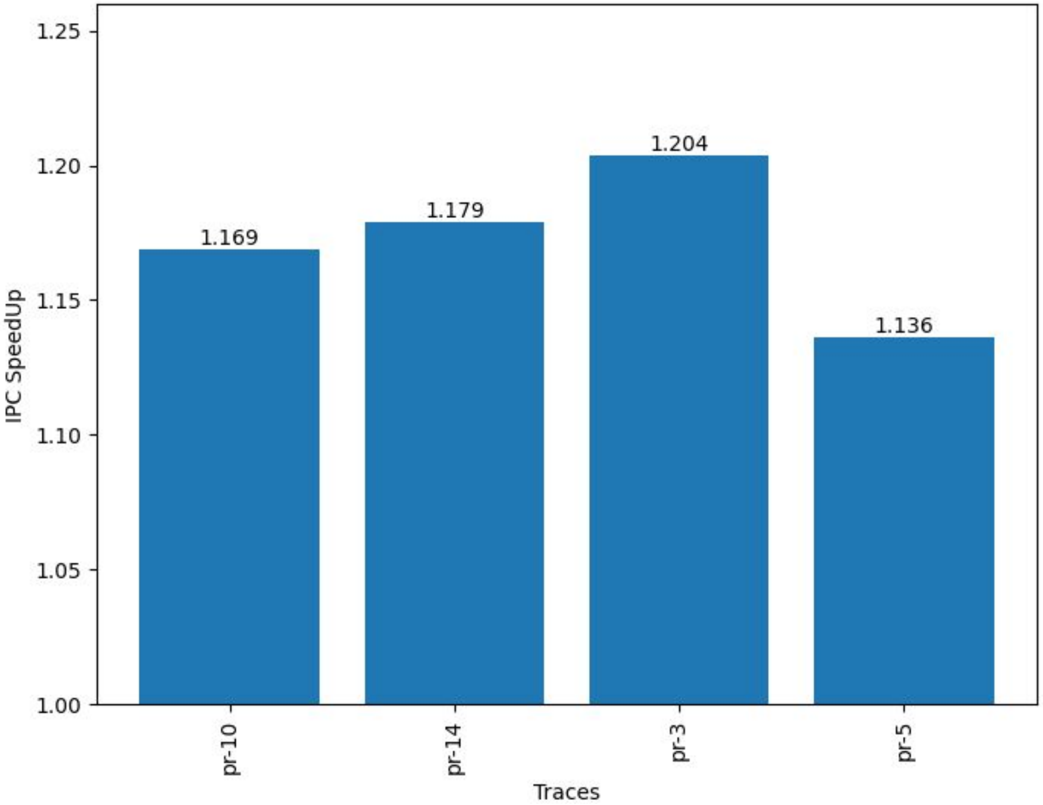
- The previous two changes convinced us that GS was **underperforming** in the **Page-Rank** traces

- So we experimented with the **priority order**

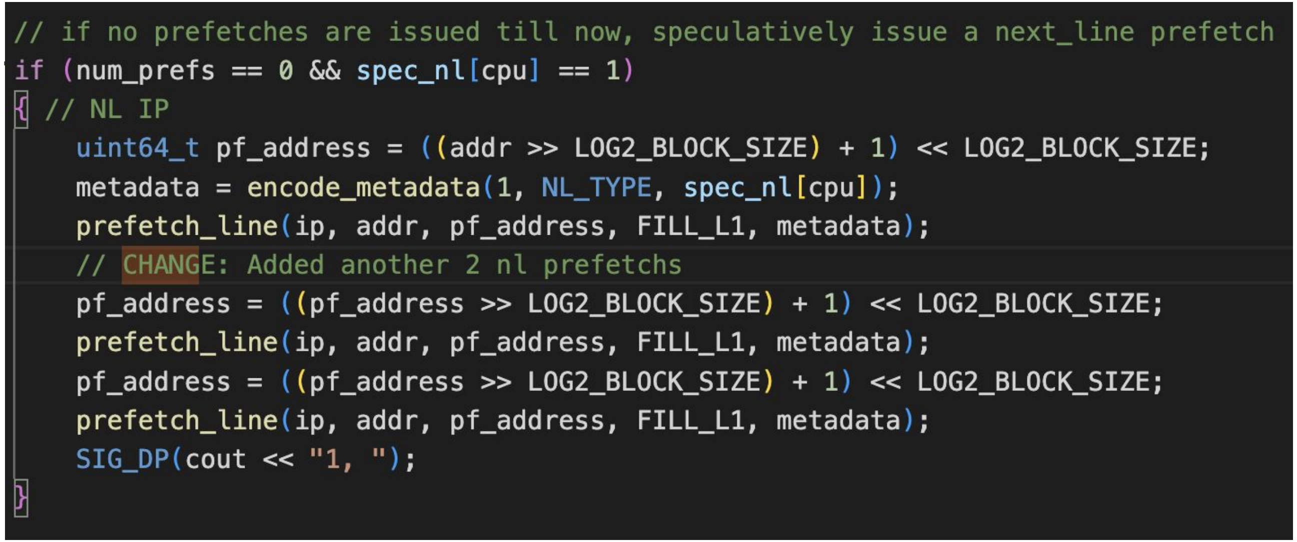
- The greatest performance boost was attained by the order

CS > CPLX > GS > NL

4) POSITIVE CHANGE: lowering GS priority



5) POSITIVE CHANGE: Varying NL prefetch degree

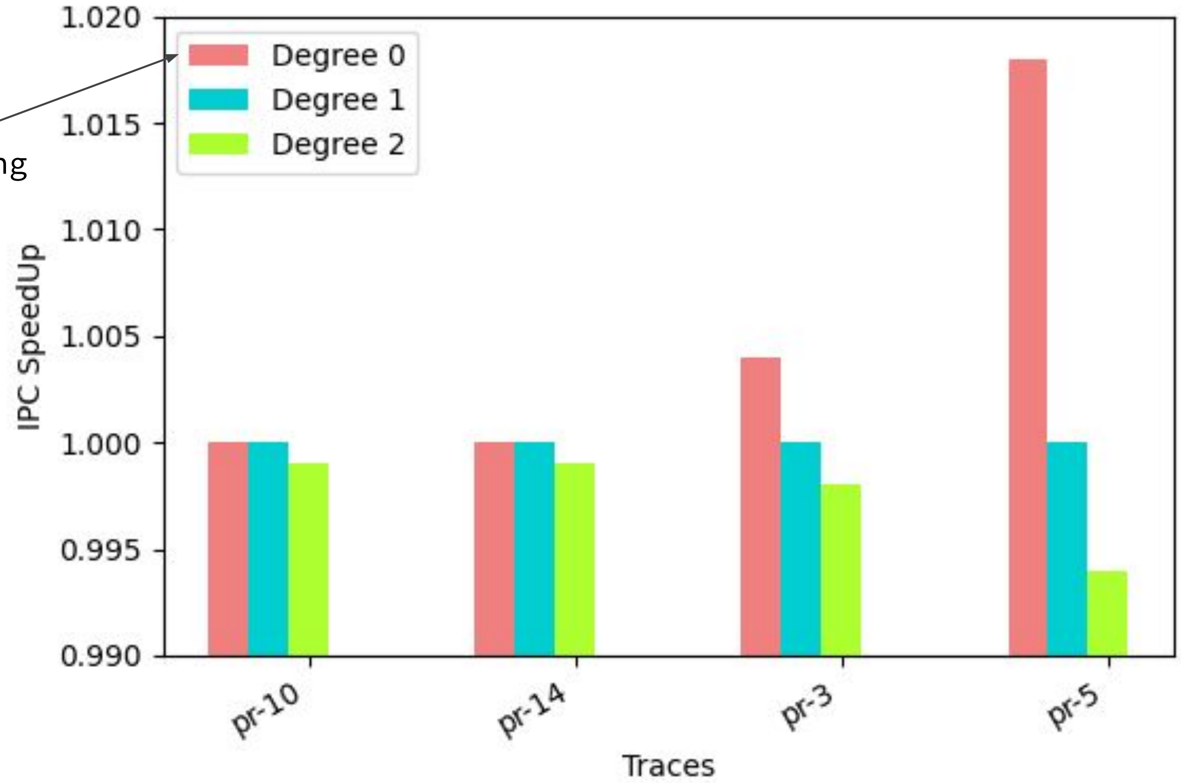


```
// if no prefetches are issued till now, speculatively issue a next_line prefetch
if (num_prefs == 0 && spec_nl[cpu] == 1)
{ // NL IP
    uint64_t pf_address = ((addr >> LOG2_BLOCK_SIZE) + 1) << LOG2_BLOCK_SIZE;
    metadata = encode_metadata(1, NL_TYPE, spec_nl[cpu]);
    prefetch_line(ip, addr, pf_address, FILL_L1, metadata);
    // CHANGE: Added another 2 nl prefetchs
    pf_address = ((pf_address >> LOG2_BLOCK_SIZE) + 1) << LOG2_BLOCK_SIZE;
    prefetch_line(ip, addr, pf_address, FILL_L1, metadata);
    pf_address = ((pf_address >> LOG2_BLOCK_SIZE) + 1) << LOG2_BLOCK_SIZE;
    prefetch_line(ip, addr, pf_address, FILL_L1, metadata);
    SIG_DP(cout << "1, ");
}
```

- Varied NL prefetch degree from **0 to 2**
- A degree of **0** means **no prefetching**
- Maximum is at **0**

5) POSITIVE CHANGE: Varying NL prefetch degree

No NL prefetching



Our Verdict on GS and NL for GRAPH

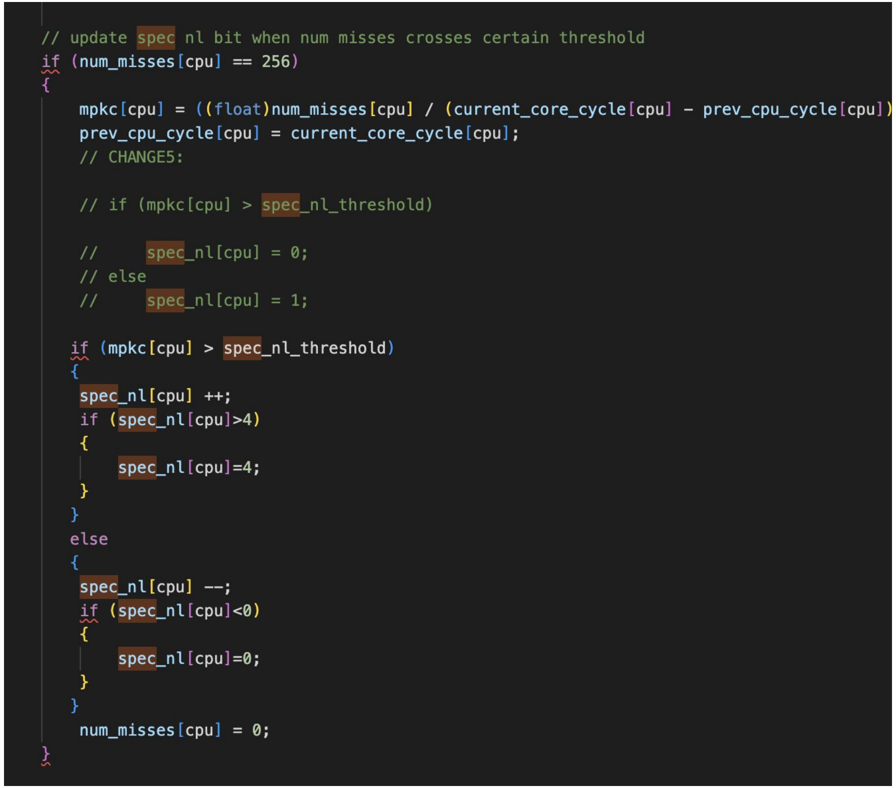
GS
CS
CPLX
NL
imgflip.com
dreamstime

6) NEUTRAL CHANGE: Throttling the whole prefetcher

- `Made spec_nl an integer instead of a bool`

Spec_nl →	0	1	2	3	4
GS	✔	✔	✔	✔	❌
CS	✔	✔	✔	❌	❌
CPLX	✔	✔	❌	❌	❌
NL	✔	❌	❌	❌	❌

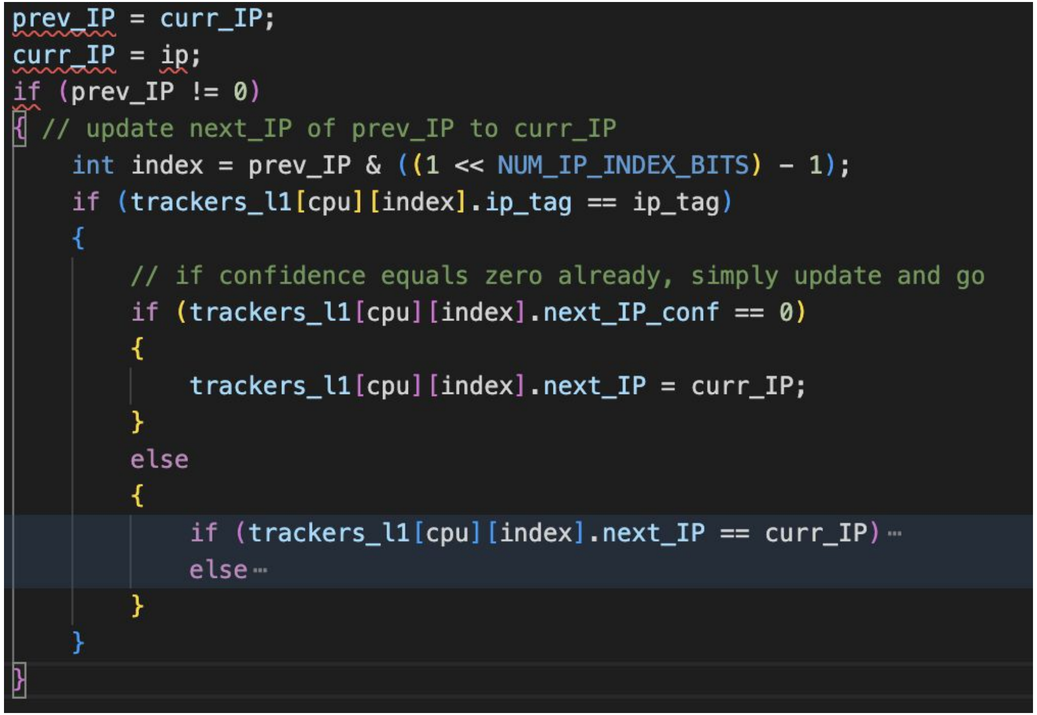
6) NEUTRAL CHANGE: Throttling the whole prefetcher



```
// update spec nl bit when num misses crosses certain threshold
if (num_misses[cpu] == 256)
{
    mpkc[cpu] = ((float)num_misses[cpu] / (current_core_cycle[cpu] - prev_cpu_cycle[cpu])
    prev_cpu_cycle[cpu] = current_core_cycle[cpu];
    // CHANGE5:

    // if (mpkc[cpu] > spec_nl_threshold)

    //      spec_nl[cpu] = 0;
    // else
    //      spec_nl[cpu] = 1;

   if (mpkc[cpu] > spec_nl_threshold)
   {
    spec_nl[cpu] ++;
    if (spec_nl[cpu]>4)
    {
        spec_nl[cpu]=4;
    }
   }
   else
   {
    spec_nl[cpu] --;
    if (spec_nl[cpu]<0)
    {
        spec_nl[cpu]=0;
    }
   }
    num_misses[cpu] = 0;
}
```

- IPCP just throttles NL prefetching
- nawaabFetcher **progressively throttles** all prefetching when **MPKC** is **continuously high**
- **No noticeable improvement**

7) NEUTRAL CHANGE: Context-aware Prefetching

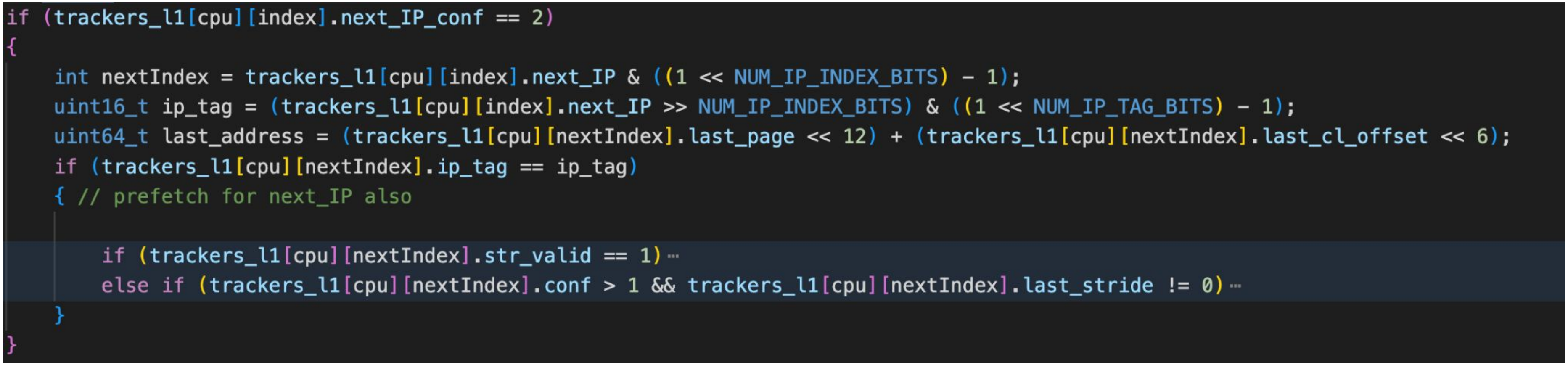


```
prev_IP = curr_IP;
curr_IP = ip;
if (prev_IP != 0)
{ // update next_IP of prev_IP to curr_IP
    int index = prev_IP & ((1 << NUM_IP_INDEX_BITS) - 1);
    if (trackers_l1[cpu][index].ip_tag == ip_tag)
    {
        // if confidence equals zero already, simply update and go
        if (trackers_l1[cpu][index].next_IP_conf == 0)
        {
            trackers_l1[cpu][index].next_IP = curr_IP;
        }
        else
        {
            if (trackers_l1[cpu][index].next_IP == curr_IP) ⋯
            else ⋯
        }
    }
}
```

- Added **two more columns** to IP_TABLE: **next_IP** and **next_IP_conf**

- When an IP is **repeatedly followed** by the same IP, **increment confidence**

8) NEUTRAL CHANGE: Context-aware Prefetching



```
if (trackers_l1[cpu][index].next_IP_conf == 2)
{
    int nextIndex = trackers_l1[cpu][index].next_IP & ((1 << NUM_IP_INDEX_BITS) - 1);
    uint16_t ip_tag = (trackers_l1[cpu][index].next_IP >> NUM_IP_INDEX_BITS) & ((1 << NUM_IP_TAG_BITS) - 1);
    uint64_t last_address = (trackers_l1[cpu][nextIndex].last_page << 12) + (trackers_l1[cpu][nextIndex].last_cl_offset << 6);
    if (trackers_l1[cpu][nextIndex].ip_tag == ip_tag)
    { // prefetch for next_IP also

        if (trackers_l1[cpu][nextIndex].str_valid == 1)⋯
        else if (trackers_l1[cpu][nextIndex].conf > 1 && trackers_l1[cpu][nextIndex].last_stride != 0)⋯
    }
}
```

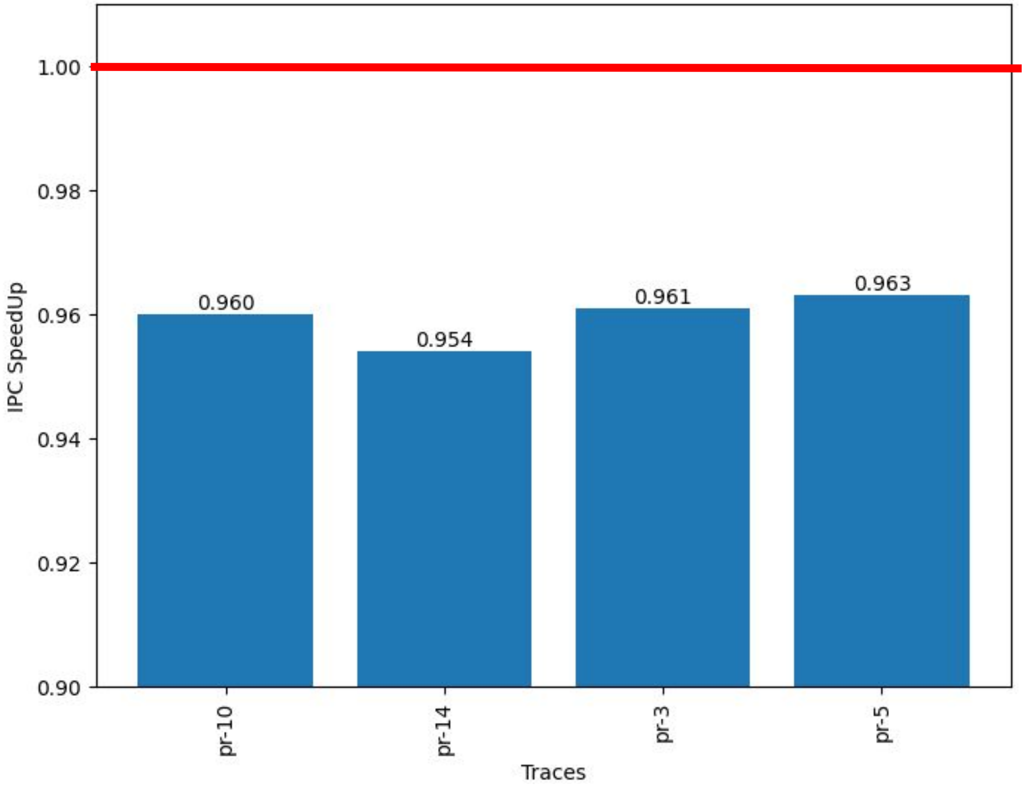
- If **next_IP_conf** is **high** (==2), **guess the next_IP's access** based on its IP_table entry and prefetch addresses for next_IP **in advance**

- **Only implementable** for **GS** and **CS** classes, as next_IP's access is **easily guessable** for GS and CS

9) NEGATIVE CHANGE: LLC IMPLEMENTATION SIMILAR TO L2C

- IPCP doesn’t implement LLC prefetching
- nawaabFetcher’s L2C **passes** the **L1D metadata to LLC**
- This enables LLC to maintain a **prefetcher state** with a **similar structure to L2C**, and prefetch similarly
- This resulted in a staggeringly **low LLC prefetch accuracy** (~1%)
- We believe this is because the LLC tries to prefetch **two steps ahead** of L1D, and any **minute disturbance** in the **access pattern** throws off the LLC

9) NEGATIVE CHANGE: LLC IMPLEMENTATION SIMILAR TO L2C



10) NEUTRAL CHANGE: CPLX @ L2C

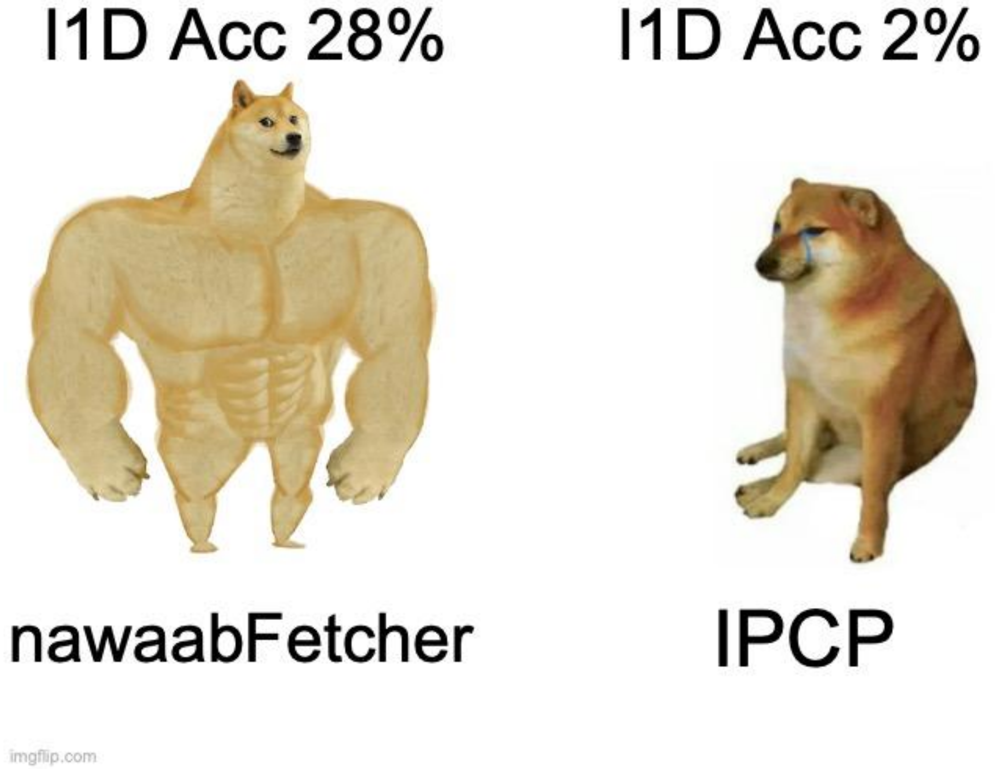
- **IPCP doesn’t** implement CPLX @ L2C due to **insignificant improvements**

- Our results **agree** with the insignificance of CPLX @ L2C. All speedups are **0.999 - 1.000**

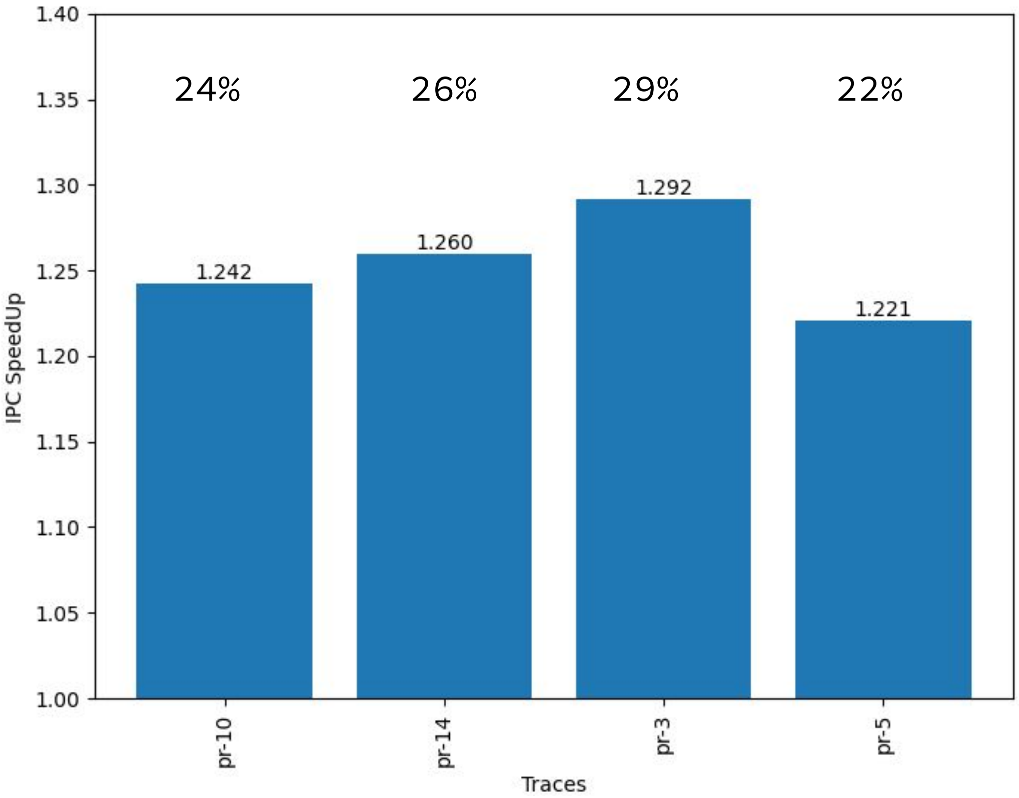
- This change suffers a **similar problem** as the change implementing the **LLC prefetcher**

- It is difficult for L2C to stay **one step ahead of L1D** due to the **unpredictability of CPLX**

Showdown !!!



All positive changes together



THANK YOU BISWA!

आपका दिन शुभ हो