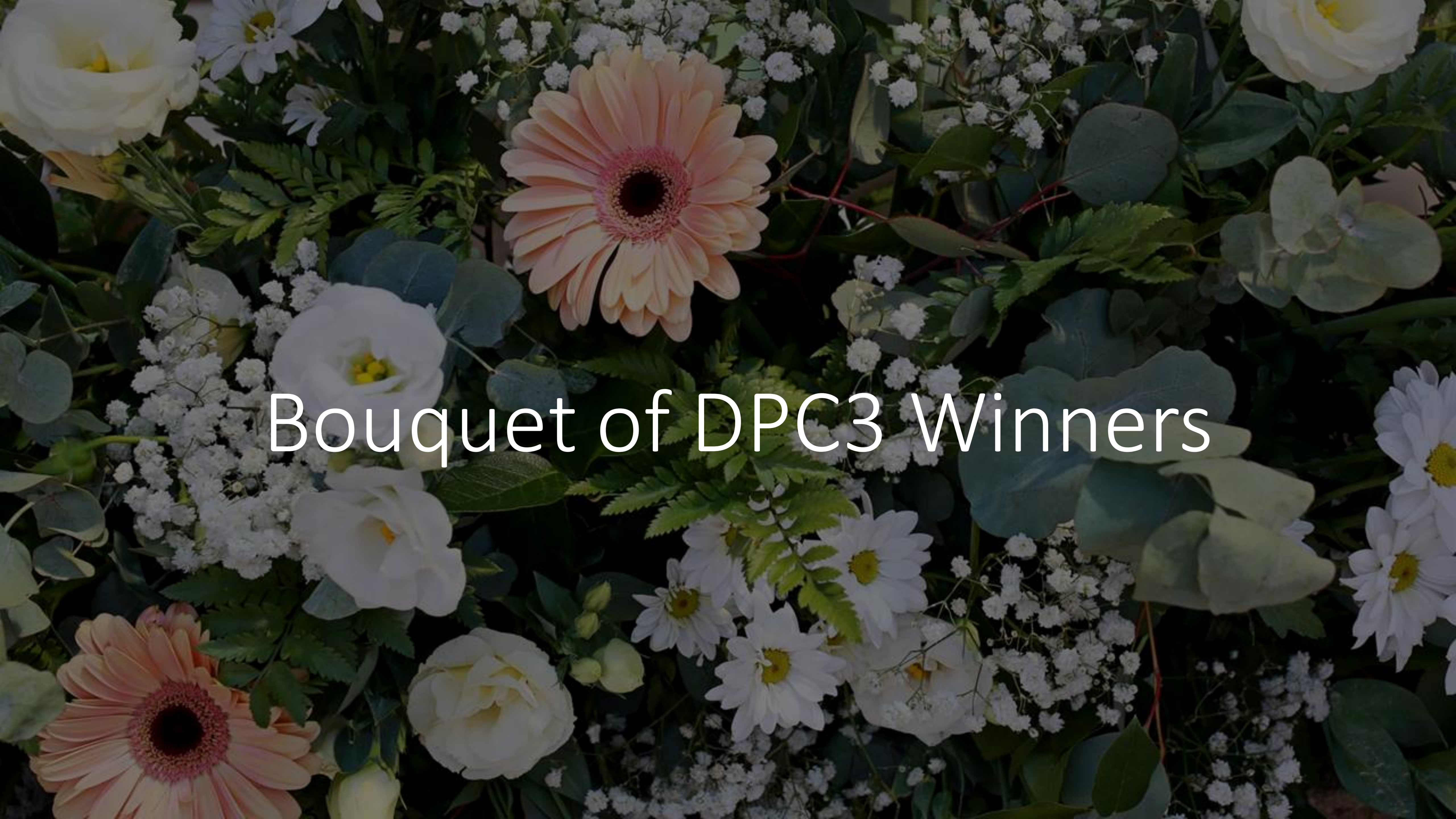
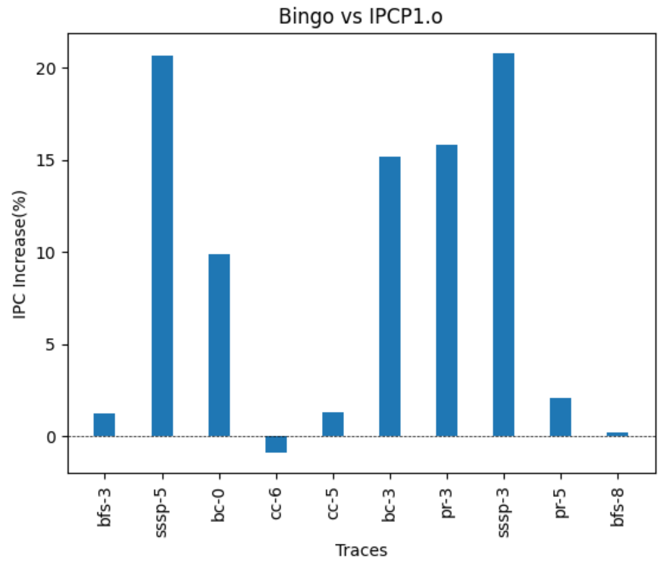


Bouquet of DPC3 Winners

Our dream? Improve the performance of the bouquet of prefetchers

Checking the plot for graph traces for the DPC3 Pros (a possibility of a perfect pairing :))

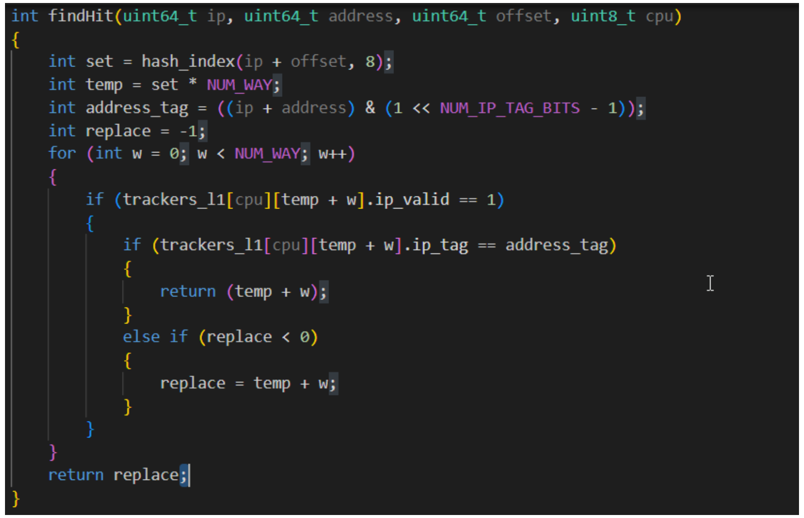


BINGO performs extremely well on graph traces

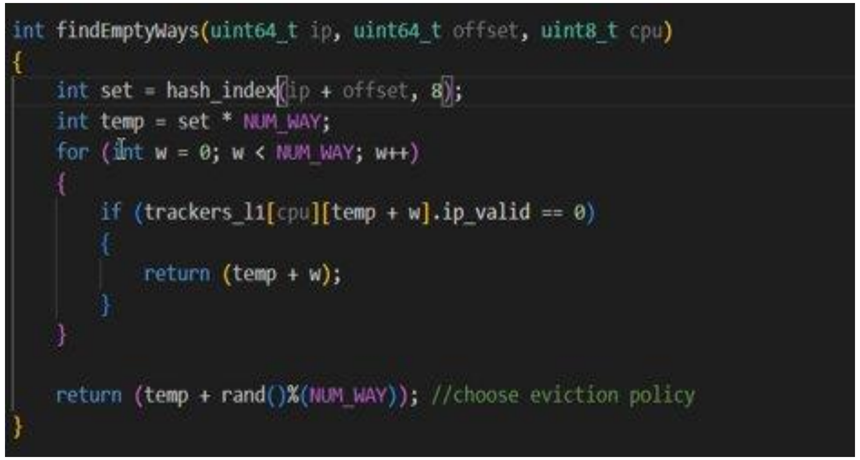
IPCP currently doesn't use associativity ideas in its tables

Opportunity to introduce this idea of BINGO into IPCP

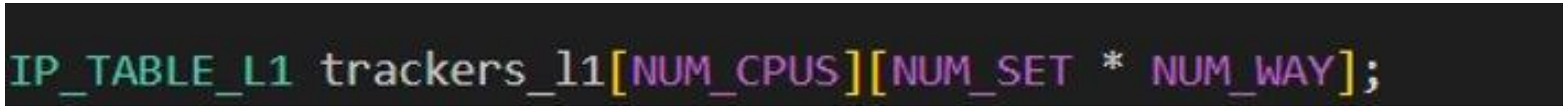
Introducing associativity in the table (kinda a cache). Does it work?



```
int findHit(uint64_t ip, uint64_t address, uint64_t offset, uint8_t cpu)
{
    int set = hash_index(ip + offset, 8);
    int temp = set * NUM_WAY;
    int address_tag = ((ip + address) & (1 << NUM_IP_TAG_BITS - 1));
    int replace = -1;
    for (int w = 0; w < NUM_WAY; w++)
    {
        if (trackers_l1[cpu][temp + w].ip_valid == 1)
        {
            if (trackers_l1[cpu][temp + w].ip_tag == address_tag)
            {
                return (temp + w);
            }
            else if (replace < 0)
            {
                replace = temp + w;
            }
        }
    }
    return replace;
}
```

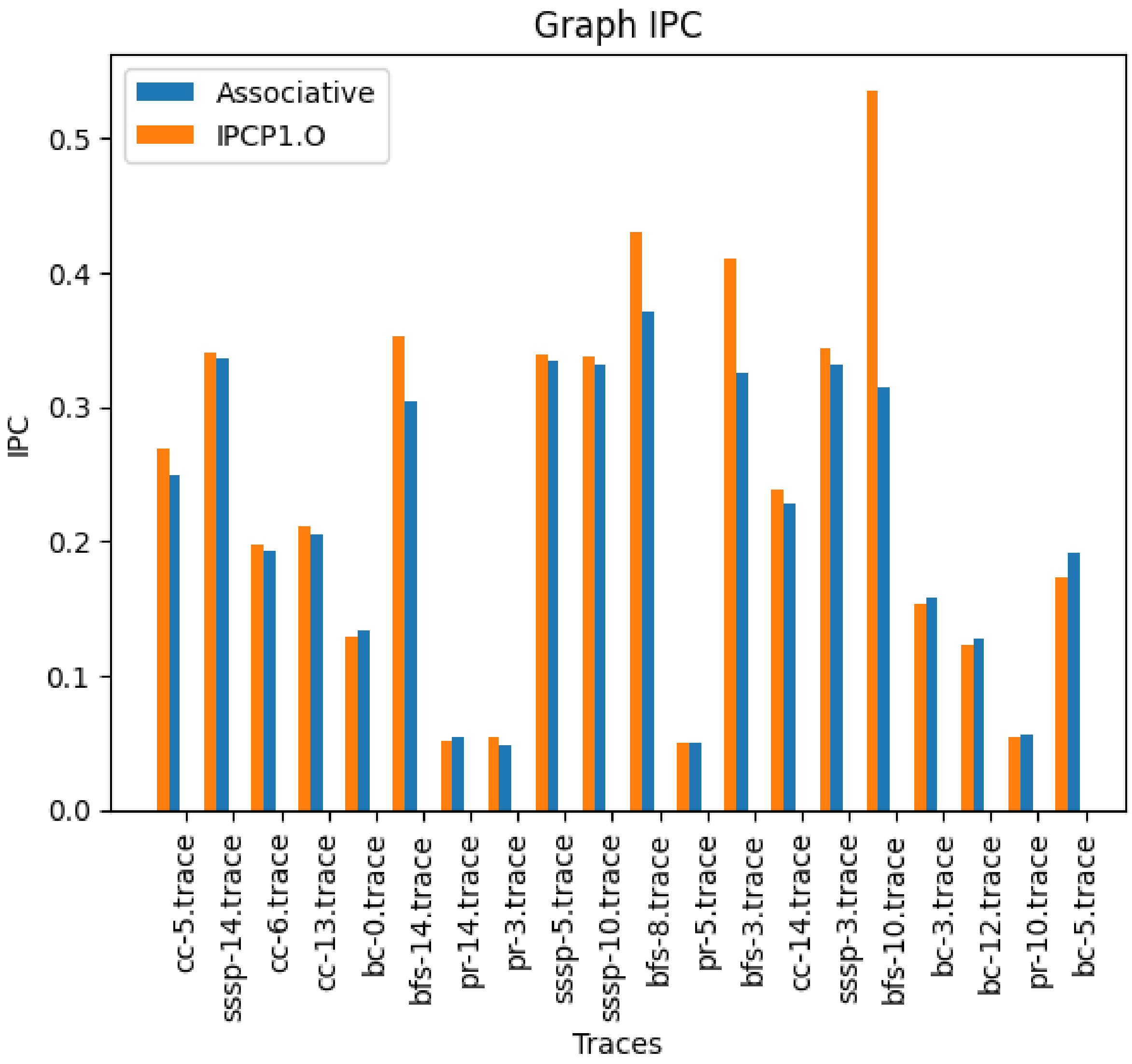


```
int findEmptyWays(uint64_t ip, uint64_t offset, uint8_t cpu)
{
    int set = hash_index(ip + offset, 8);
    int temp = set * NUM_WAY;
    for (int w = 0; w < NUM_WAY; w++)
    {
        if (trackers_l1[cpu][temp + w].ip_valid == 0)
        {
            return (temp + w);
        }
    }

    return (temp + rand()%(NUM_WAY)); //choose eviction policy
}
```



```
IP_TABLE_L1 trackers_l1[NUM_CPUS][NUM_SET * NUM_WAY];
```

Try 1 : Associative Table (kinda a cache, does not work :()

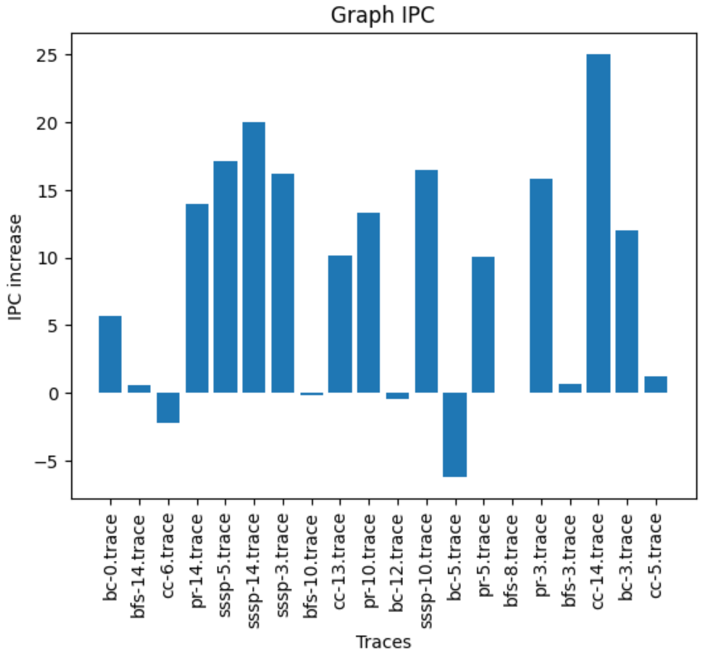


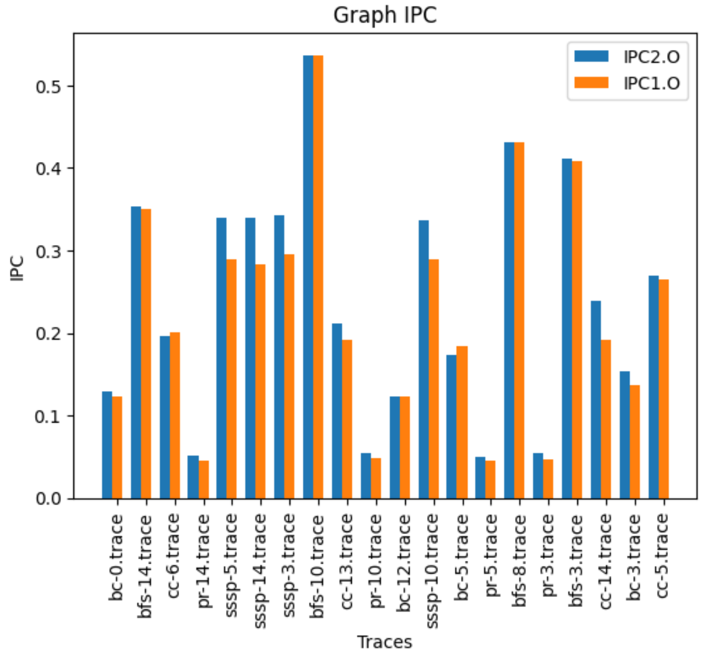
The tables add a lot of overhead

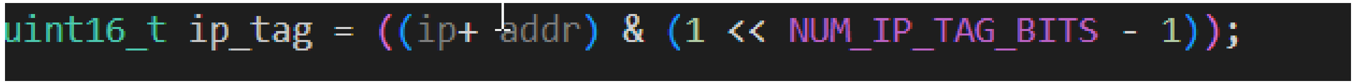
Maybe the access times and the complicated replacement schemes are adding latency?

Simplification while conserving the core ideas of the BINGO prefetcher can help! (best of both worlds)

Try 2 : Separating the tagging and indexing policies :)







```
uint16_t ip_tag = ((ip+ addr) & (1 << NUM_IP_TAG_BITS - 1));
```

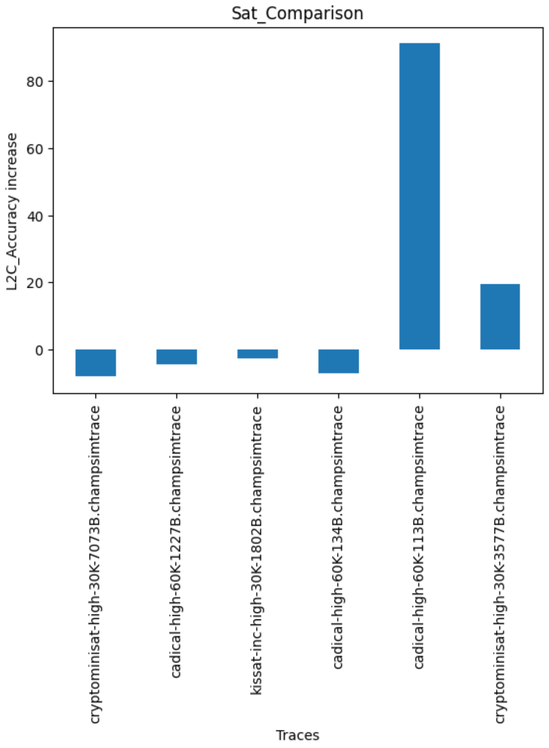
Seems to be a good result

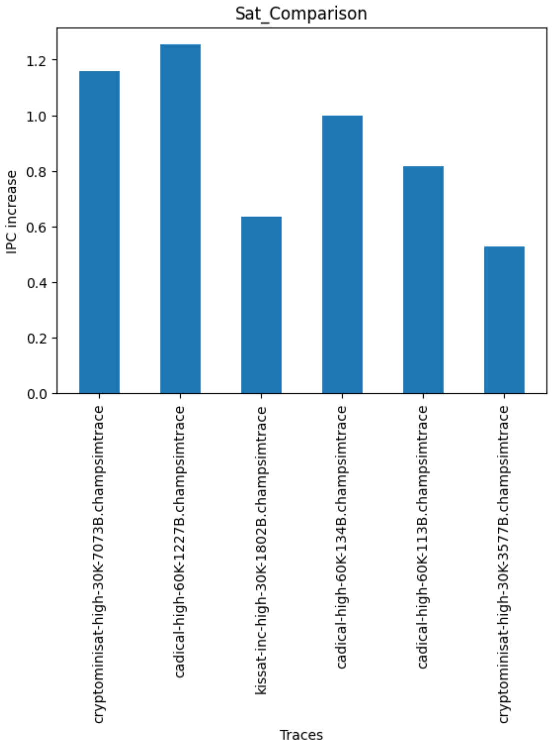
We do need to check it on other traces though. Architecture is all about tradeoffs :)

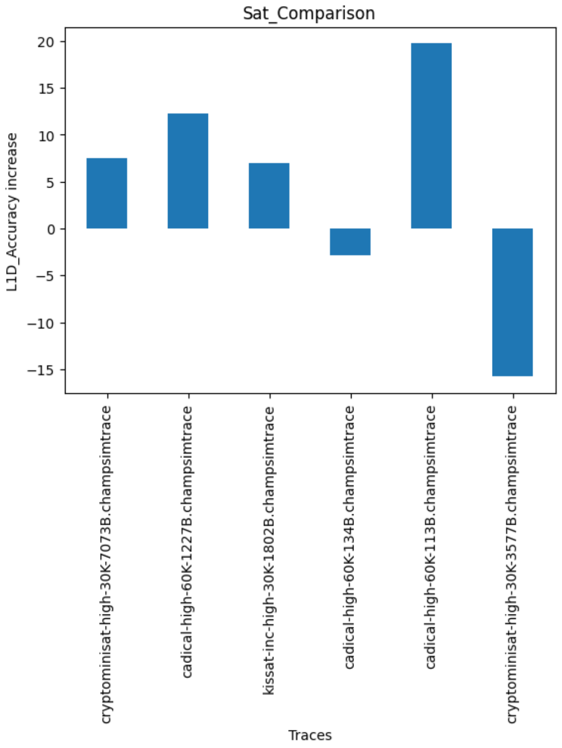
Is this a good scheme? Depends. We need to evaluate it on more traces.

Us after these results on graphs

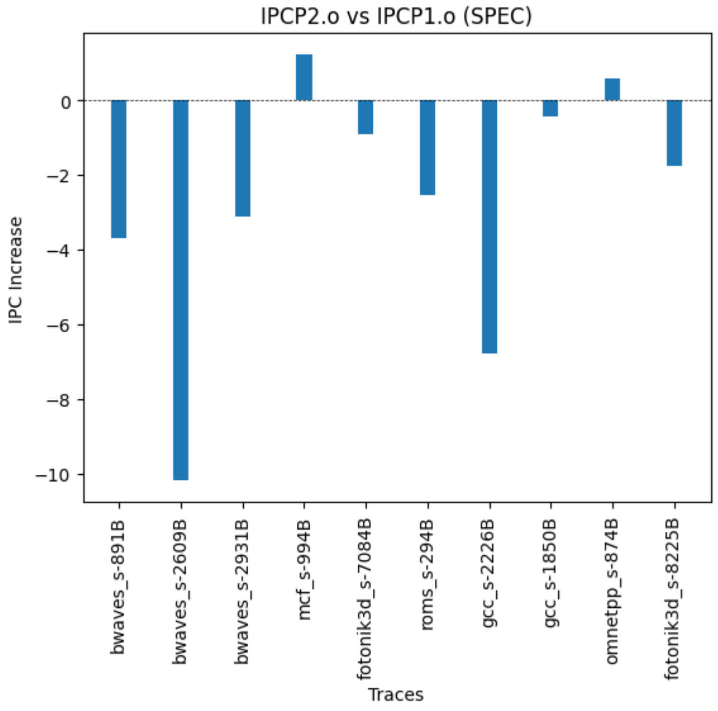
But wait! Let's improve SAT Solver stuff. Use next line in LLC. (Yay!)





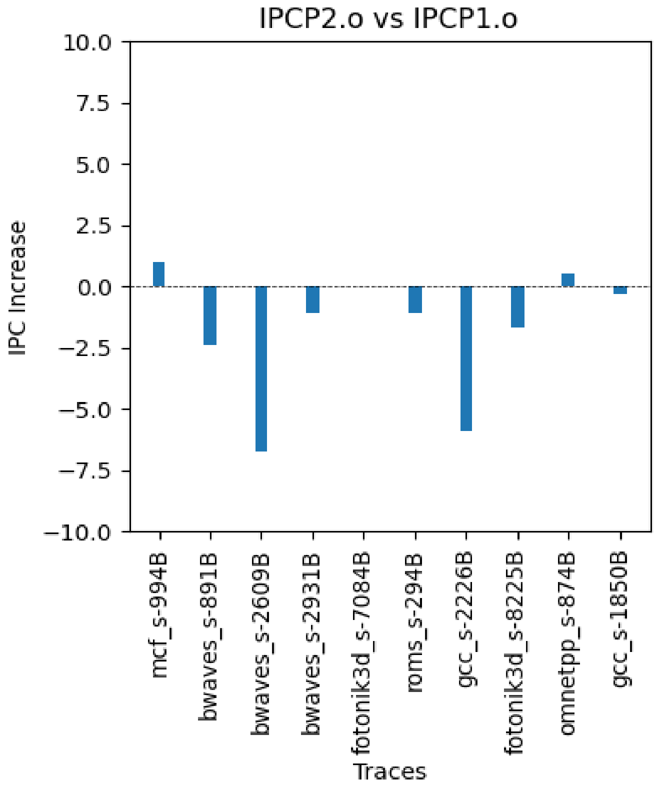


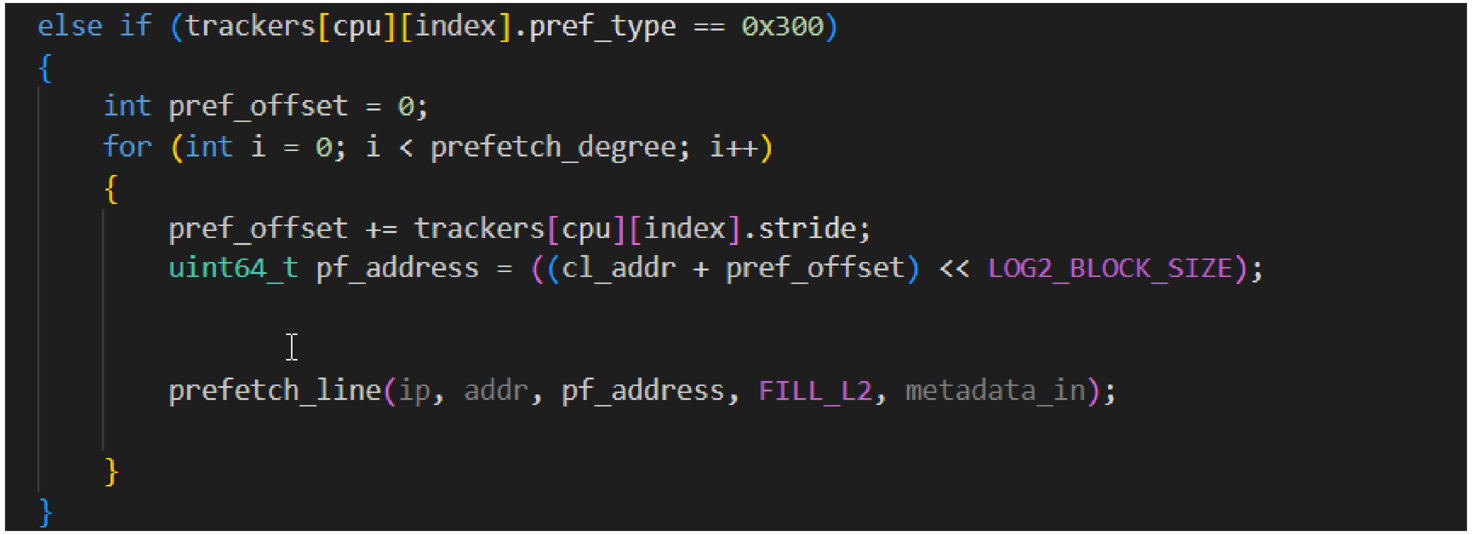
This feels weird, is a tradeoff coming? SPEC :(



Huh, a need to limit our losses and match the performance with IPCP on SPEC

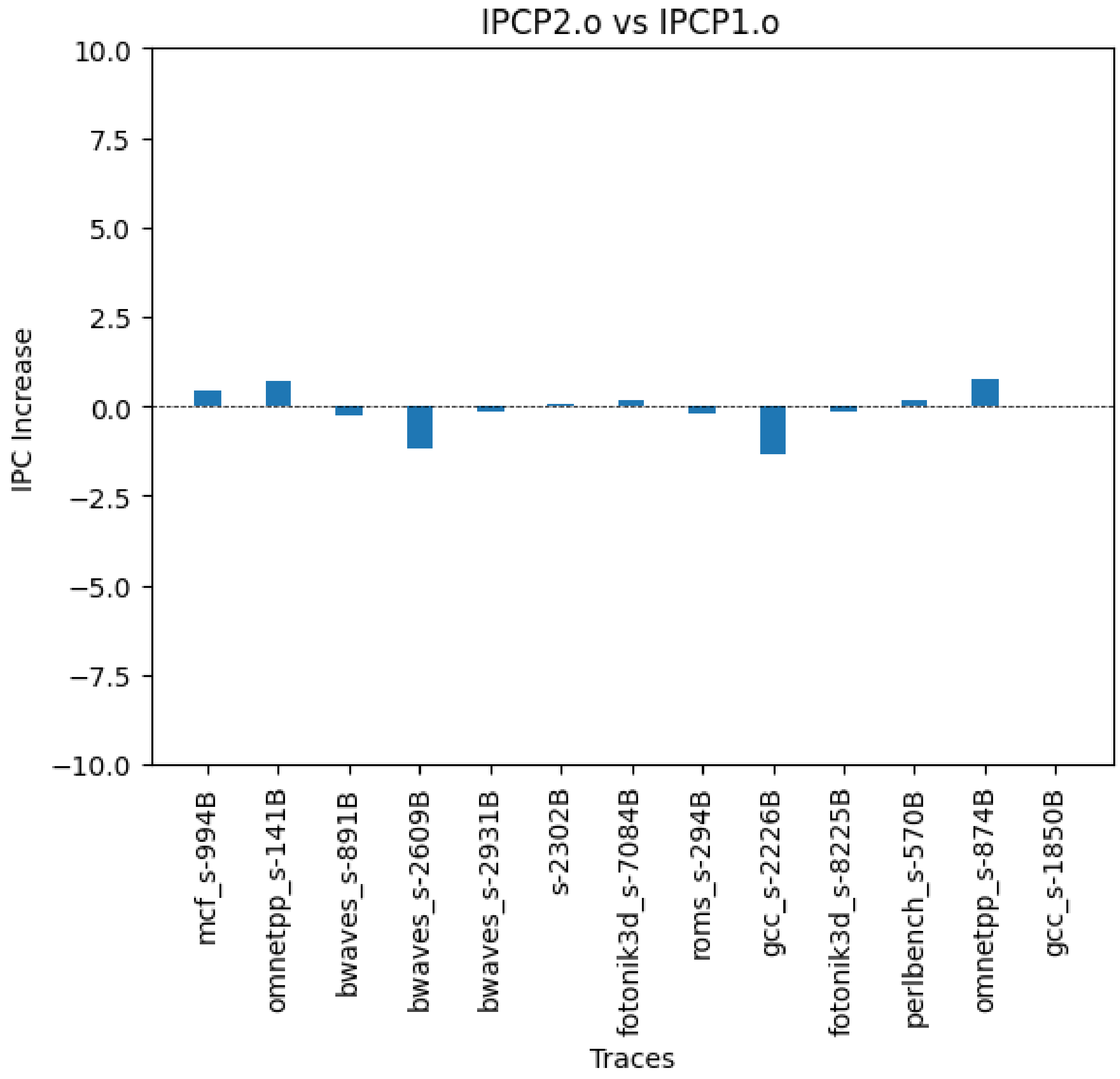
Try 1 : Integrating complex stride at L2 cache

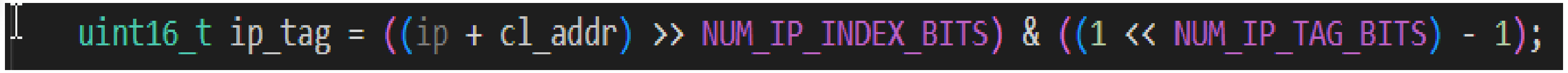




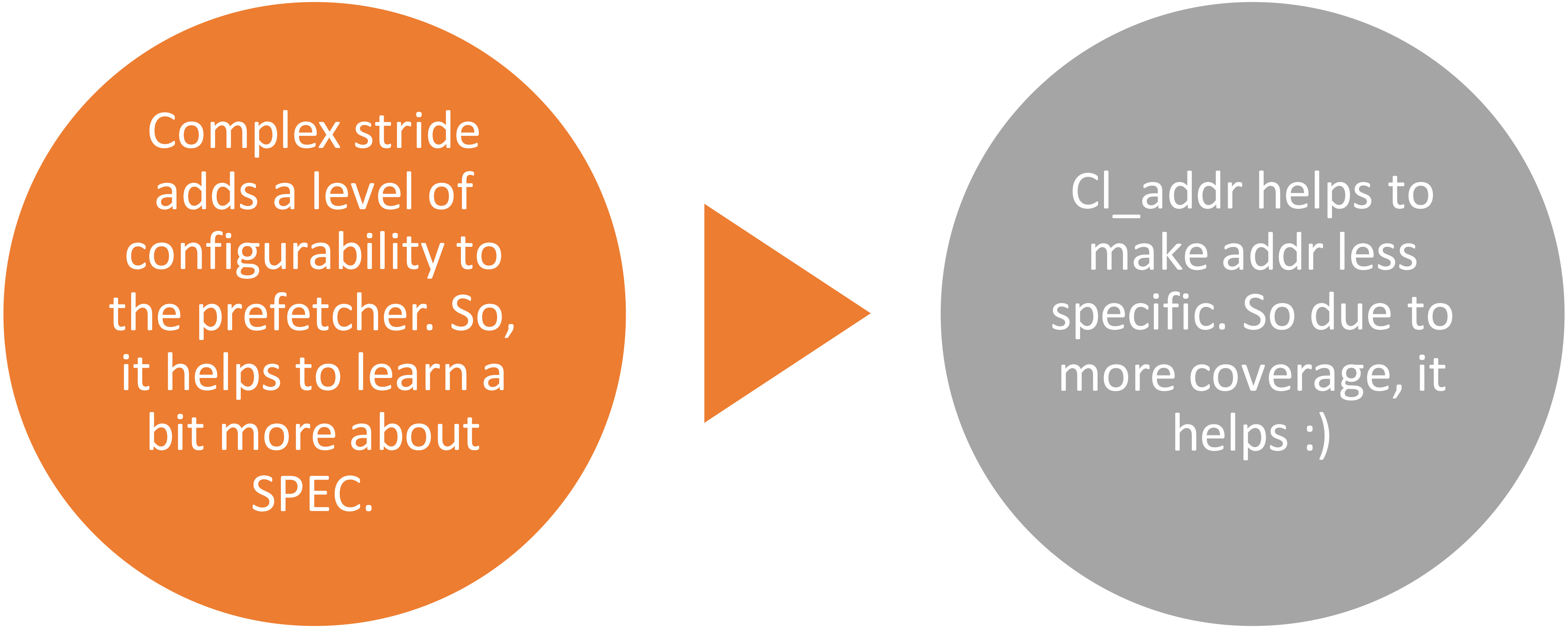
```
else if (trackers[cpu][index].pref_type == 0x300)
{
    int pref_offset = 0;
    for (int i = 0; i < prefetch_degree; i++)
    {
        pref_offset += trackers[cpu][index].stride;
        uint64_t pf_address = ((cl_addr + pref_offset) << LOG2_BLOCK_SIZE);


        prefetch_line(ip, addr, pf_address, FILL_L2, metadata_in);

    }
}
```

(Comeback!) Modifying the tag using cl_addr instead :)





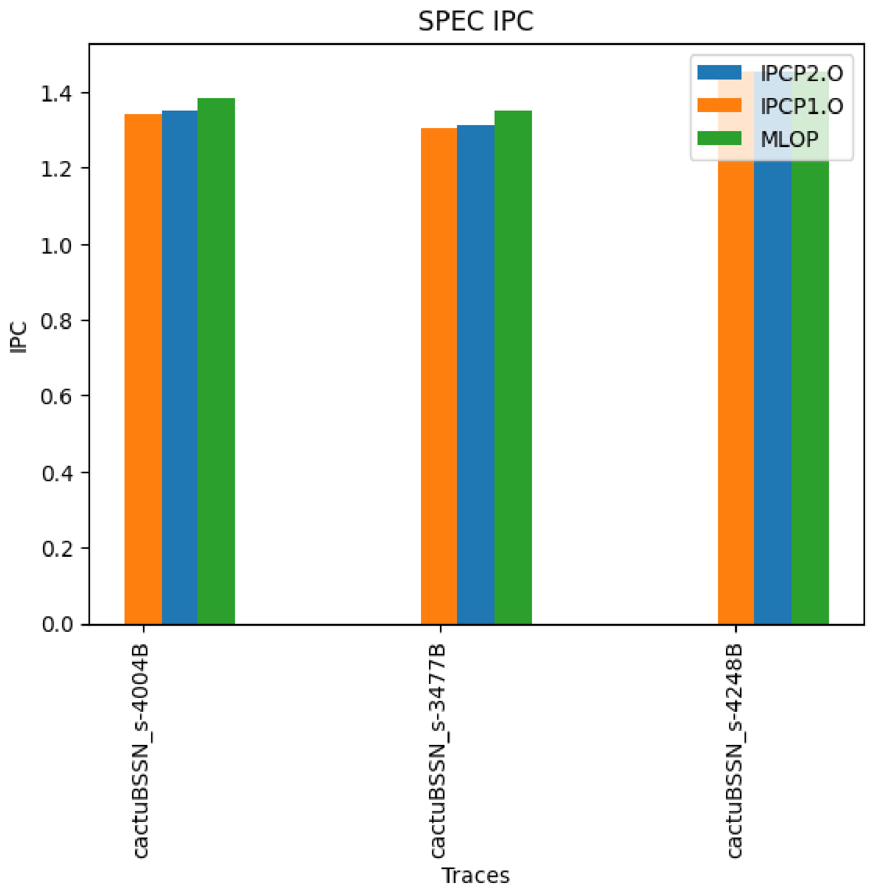
```
    uint16_t ip_tag = ((ip + cl_addr) >> NUM_IP_INDEX_BITS) & ((1 << NUM_IP_TAG_BITS) - 1);
```

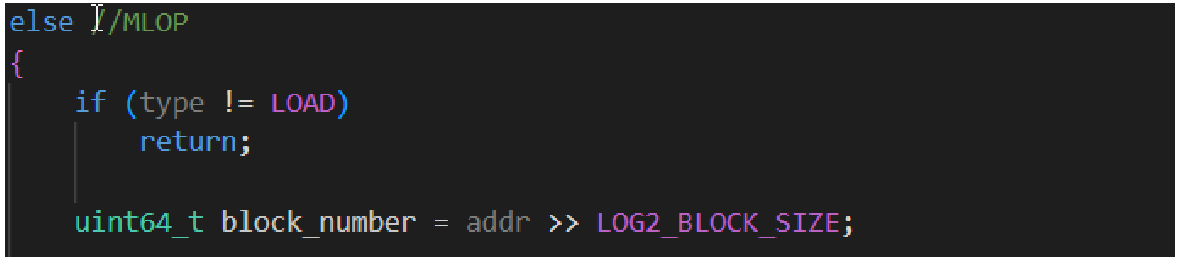


Complex stride adds a level of configurability to the prefetcher. So, it helps to learn a bit more about SPEC.

Cl_addr helps to make addr less specific. So due to more coverage, it helps :)

Integrating MLOP? Minor improvement





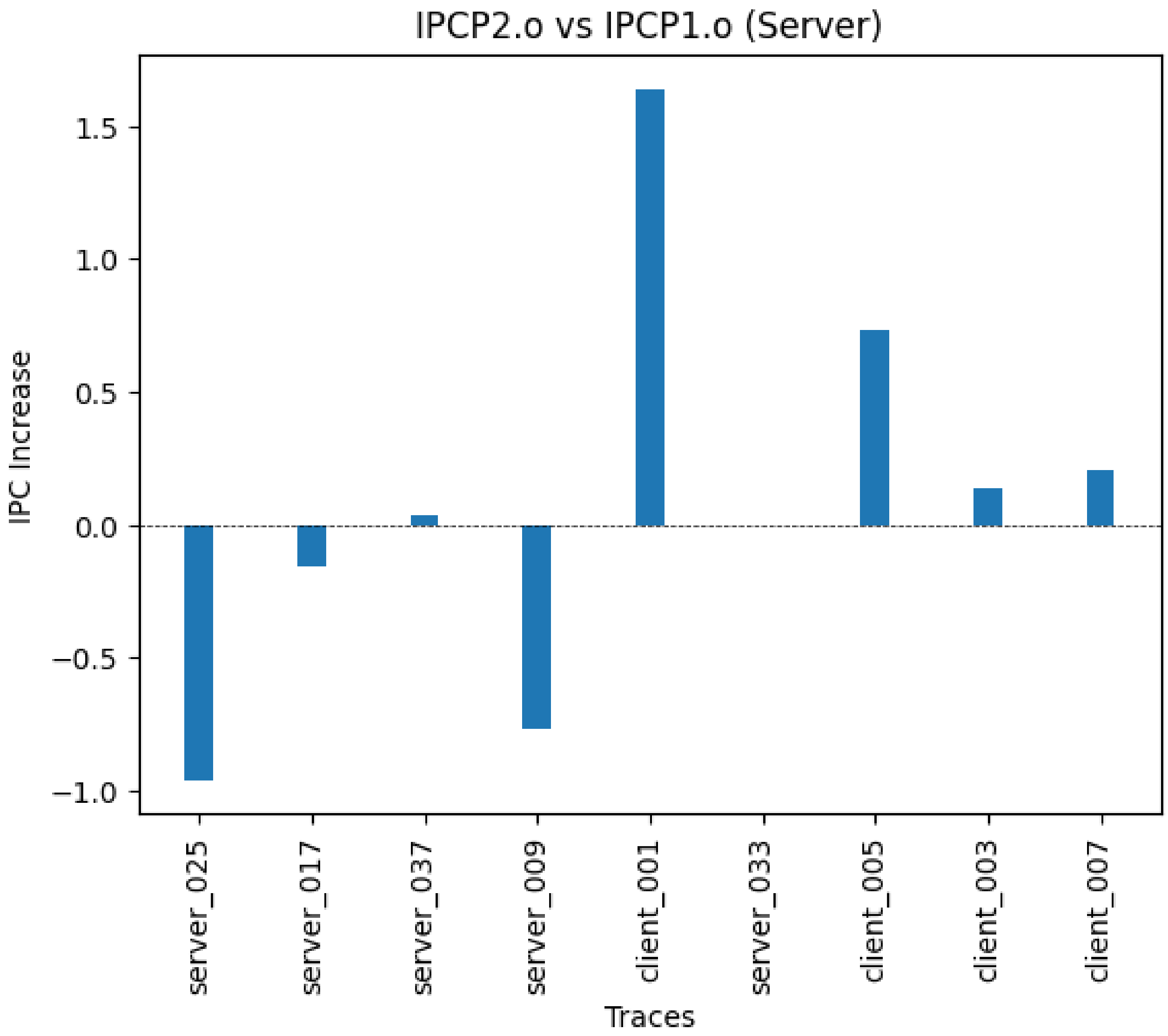
```
else //MLOP
{
    if (type != LOAD)
        return;

    uint64_t block_number = addr >> LOG2_BLOCK_SIZE;
```

MLOP prefetcher tends to do well on cactus traces

So, incorporating this prefetcher in L1 cache helps us to gain minor IPC improvements in cactus traces (SPEC)

IPC of other traces is not affected in this modification

Does it work decently enough for servers? Well, kinda

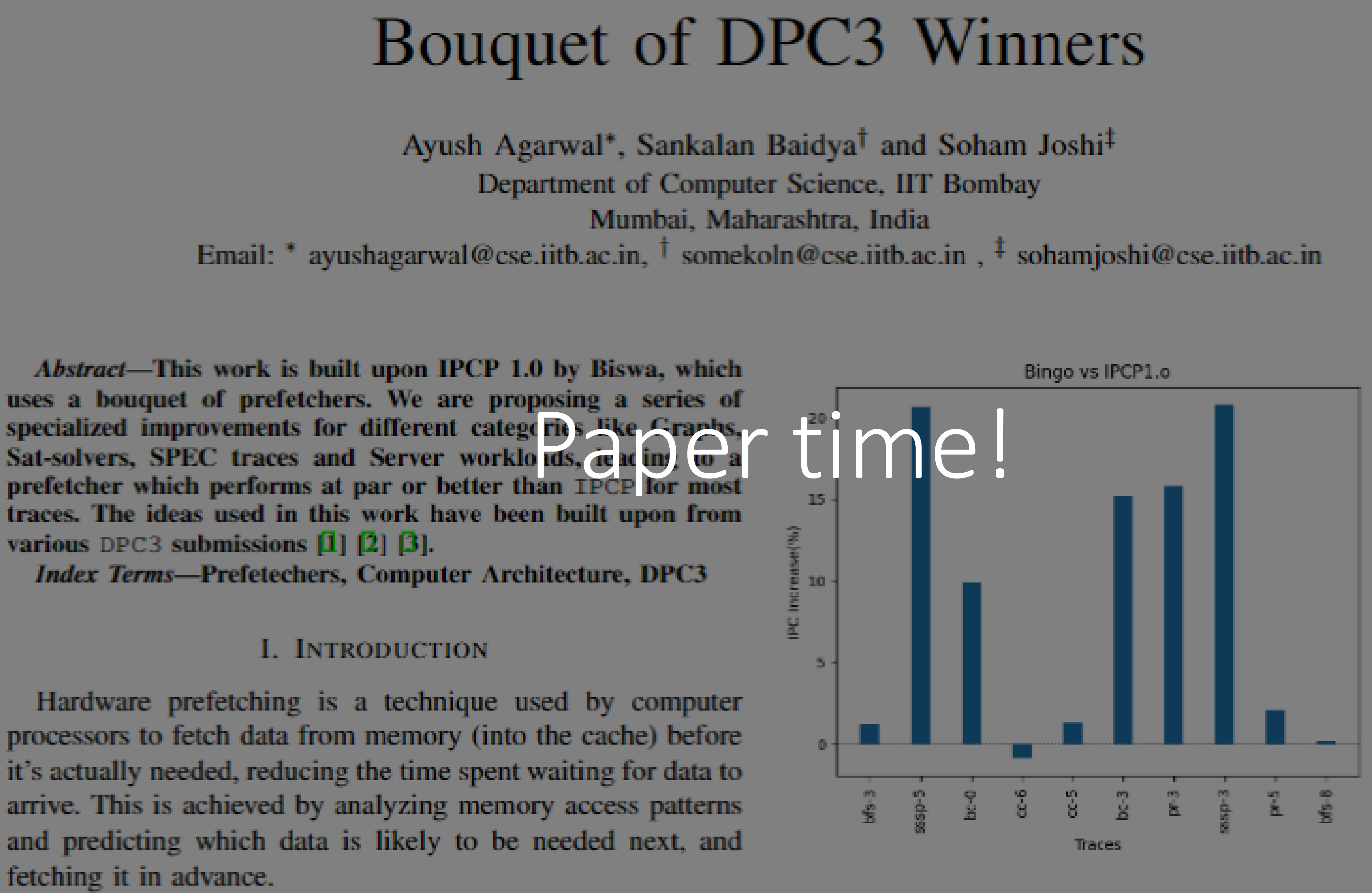


Further improvements?

Perhaps scope for further optimization?

Well, we'll leave it to another day :p

Paper time!



Bouquet of DPC3 Winners

Ayush Agarwal*, Sankalan Baidya† and Soham Joshi‡
Department of Computer Science, IIT Bombay
Mumbai, Maharashtra, India
Email: * ayushagarwal@cse.iitb.ac.in, † somekoln@cse.iitb.ac.in , ‡ sohamjoshi@cse.iitb.ac.in

***Abstract*—This work is built upon IPCP 1.0 by Biswa, which uses a bouquet of prefetchers. We are proposing a series of specialized improvements for different categories like Graphs, Sat-solvers, SPEC traces and Server workloads, leading to a prefetcher which performs at par or better than IPCP for most traces. The ideas used in this work have been built upon from various DPC3 submissions [1] [2] [3].**

***Index Terms*—Prefetechers, Computer Architecture, DPC3**

I. Introduction

Hardware prefetching is a technique used by computer processors to fetch data from memory (into the cache) before it's actually needed, reducing the time spent waiting for data to arrive. This is achieved by analyzing memory access patterns and predicting which data is likely to be needed next, and fetching it in advance.

ଆପଣଙ୍କର ଦିନ ମଙ୍ଗଳମୟ ହେଉ