

A close-up photograph of a bouquet of flowers. The bouquet features several large white roses, two prominent pink daisies with dark centers, and many small white baby's breath flowers. Green foliage and eucalyptus leaves are interspersed throughout the arrangement. The text is overlaid on the center of the image.

# Bouquet of DPC3 Winners

Ayush Agarwal (210050029)  
Sankalan Baidya (210050141)  
Soham Joshi (210051004)

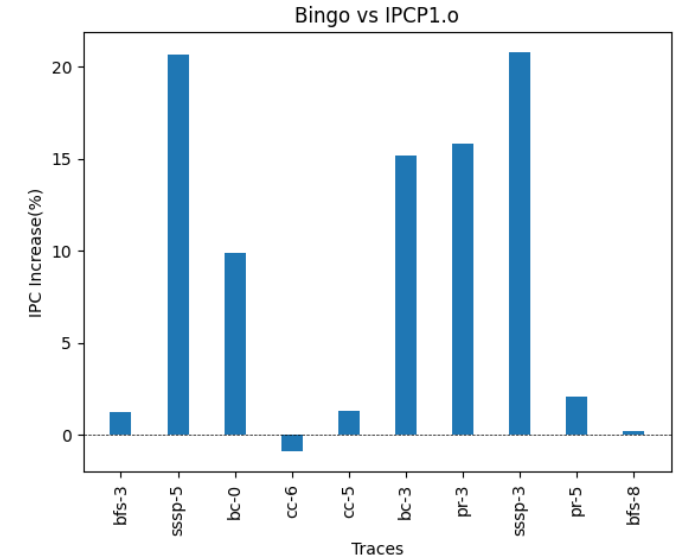
Our dream?  
Improve the  
performance of  
the bouquet of  
prefetchers

---





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?

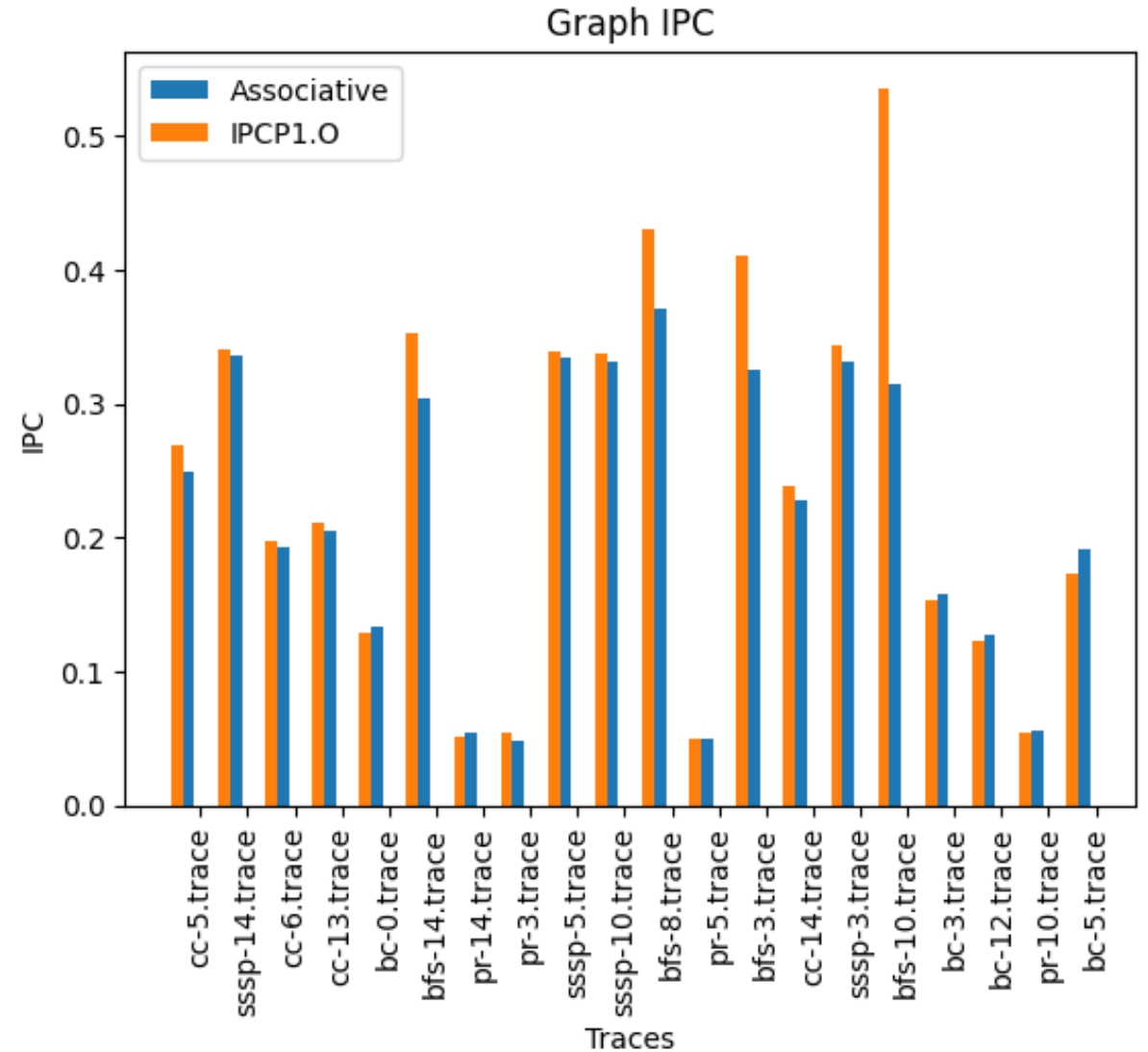
```
int findHit(uint64_t ip, uint64_t address, uint64_t offset, uint8_t cpu)
{
    int set = hash_index(ip + offset, 8);
    int temp = set * NUM_WAY;
    int address_tag = ((ip + address) & (1 << NUM_IP_TAG_BITS - 1));
    int replace = -1;
    for (int w = 0; w < NUM_WAY; w++)
    {
        if (trackers_l1[cpu][temp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

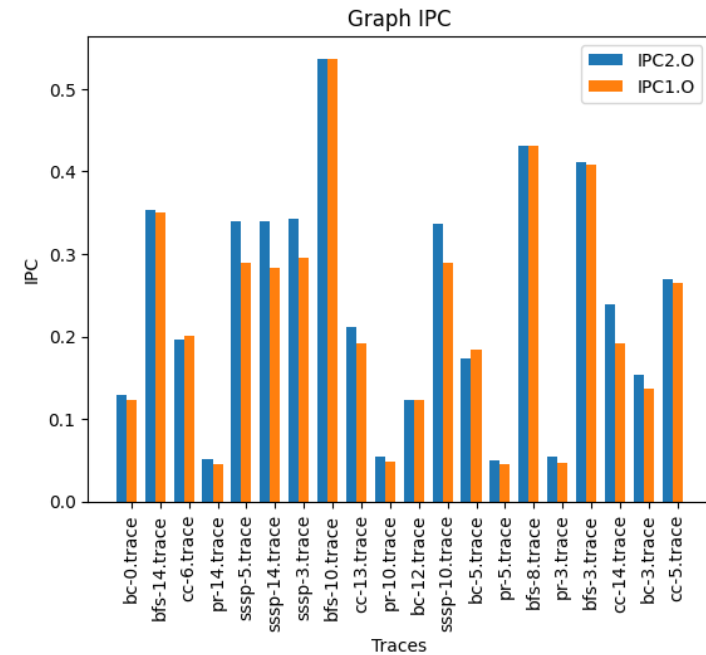
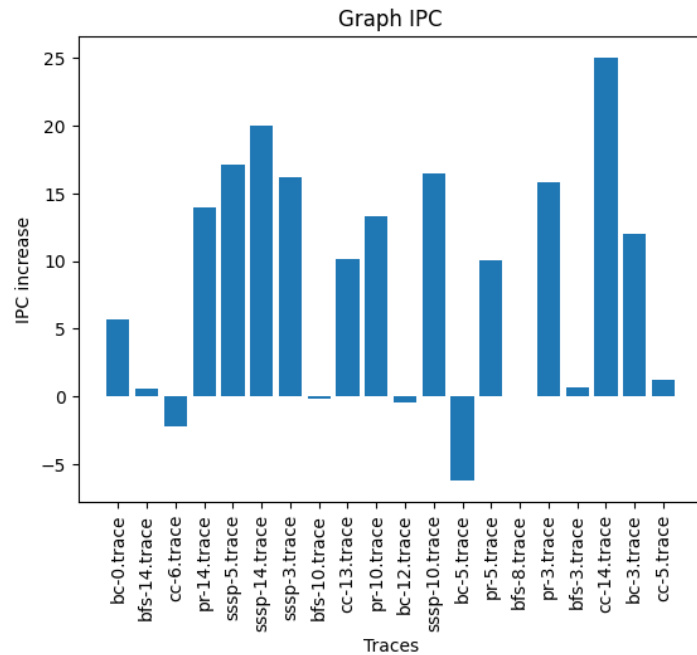


Maybe the access times and the complicated replacement schemes are adding latency?



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.

**I HAVE REACHED**

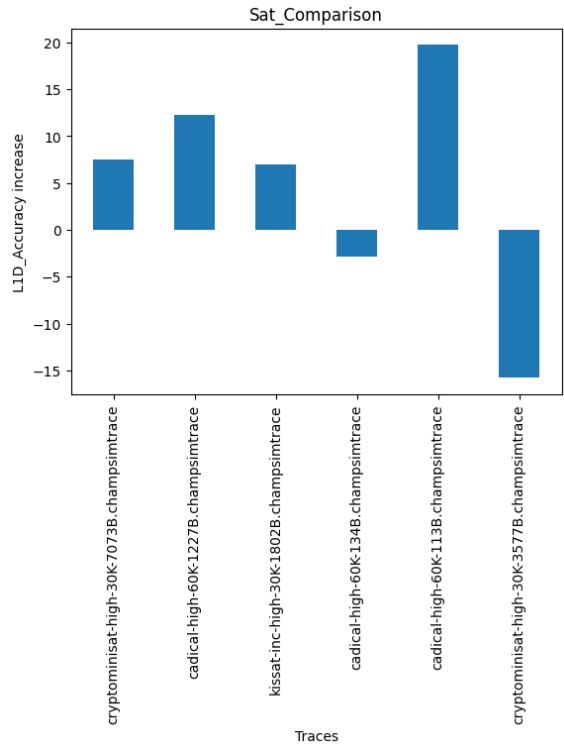
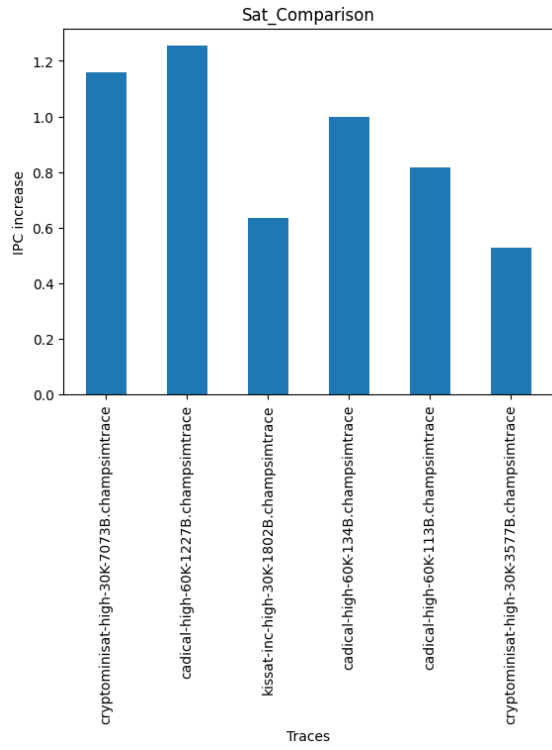
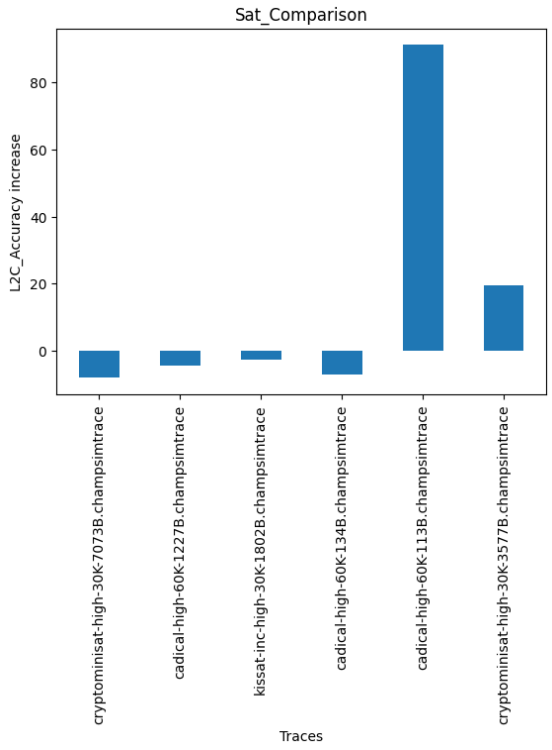


Us after  
these results  
on graphs

---

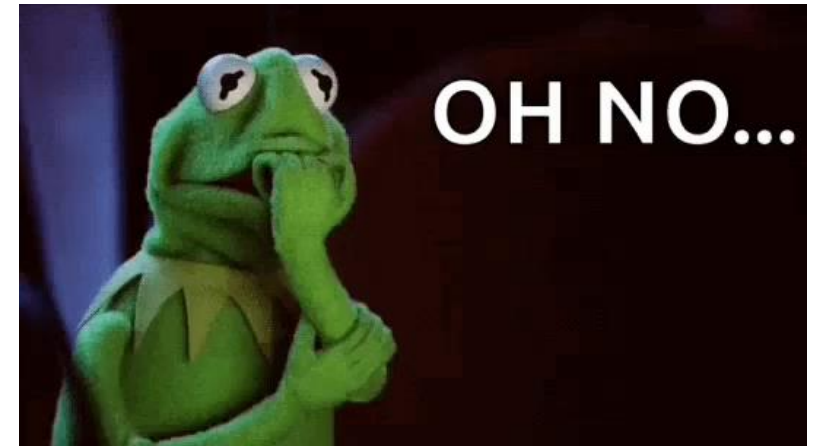
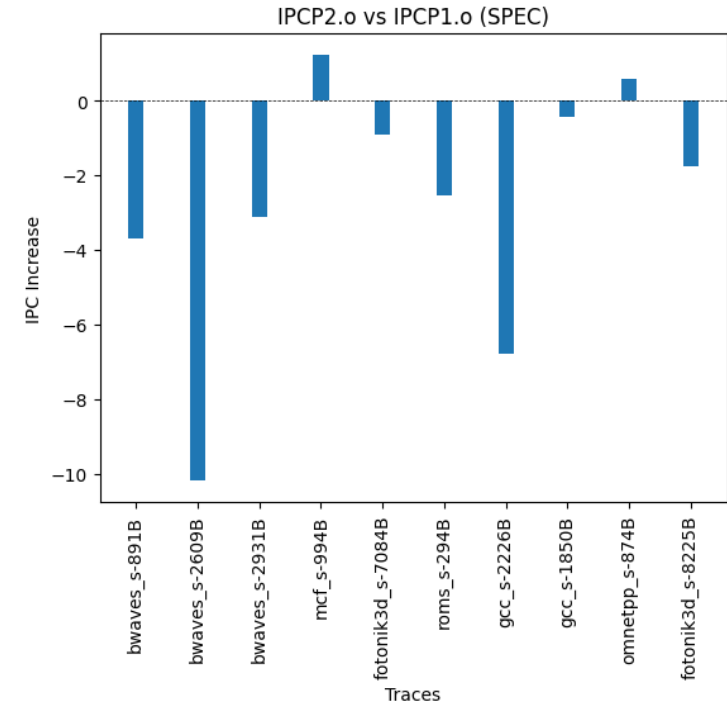
**250,000 POINTS!**

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





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



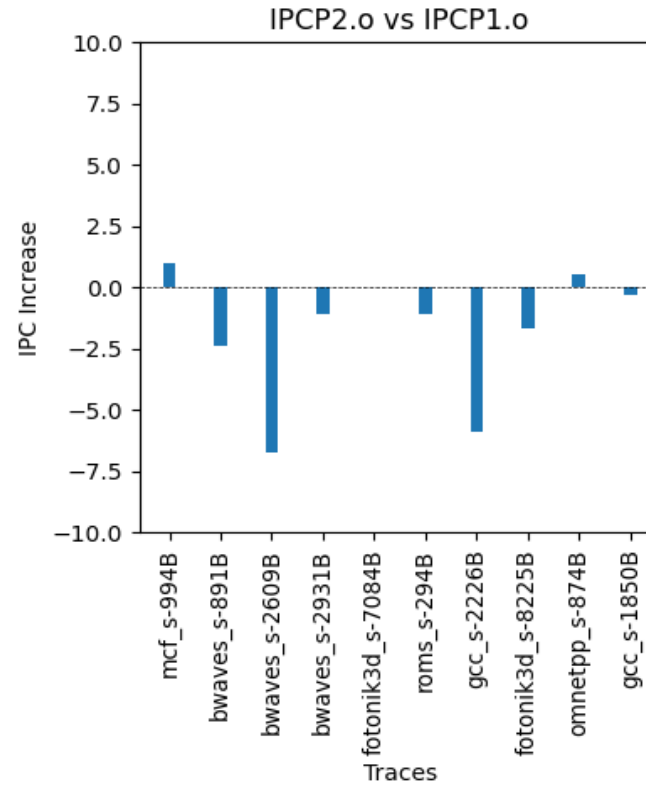
Huh, a need to  
limit our losses  
and match the  
performance  
with IPCP on  
SPEC

---

THE COMEBACK  
is always  
stronger  
than  
the setback



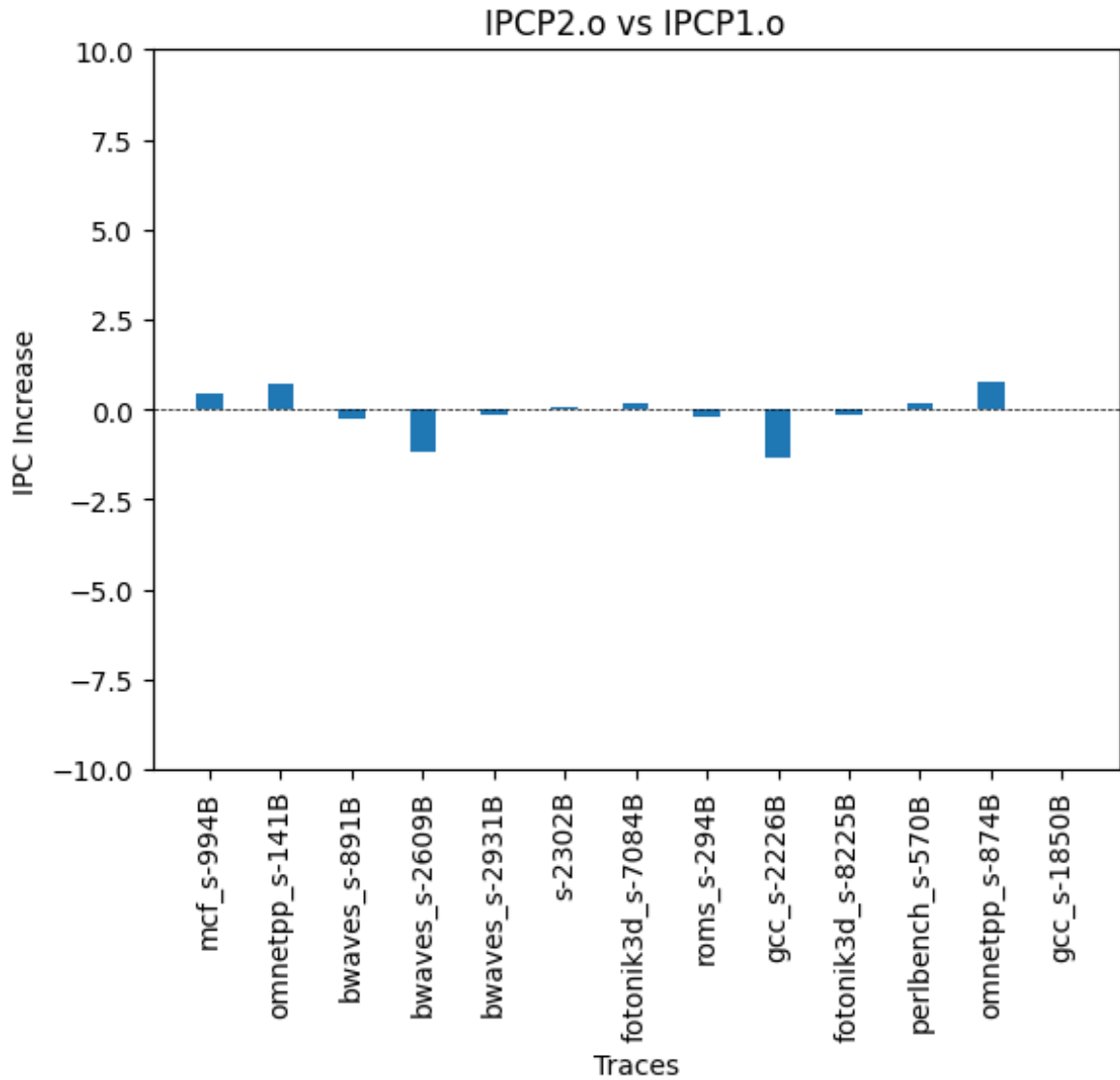
# Try 1 : Integrating complex stride at L2 cache



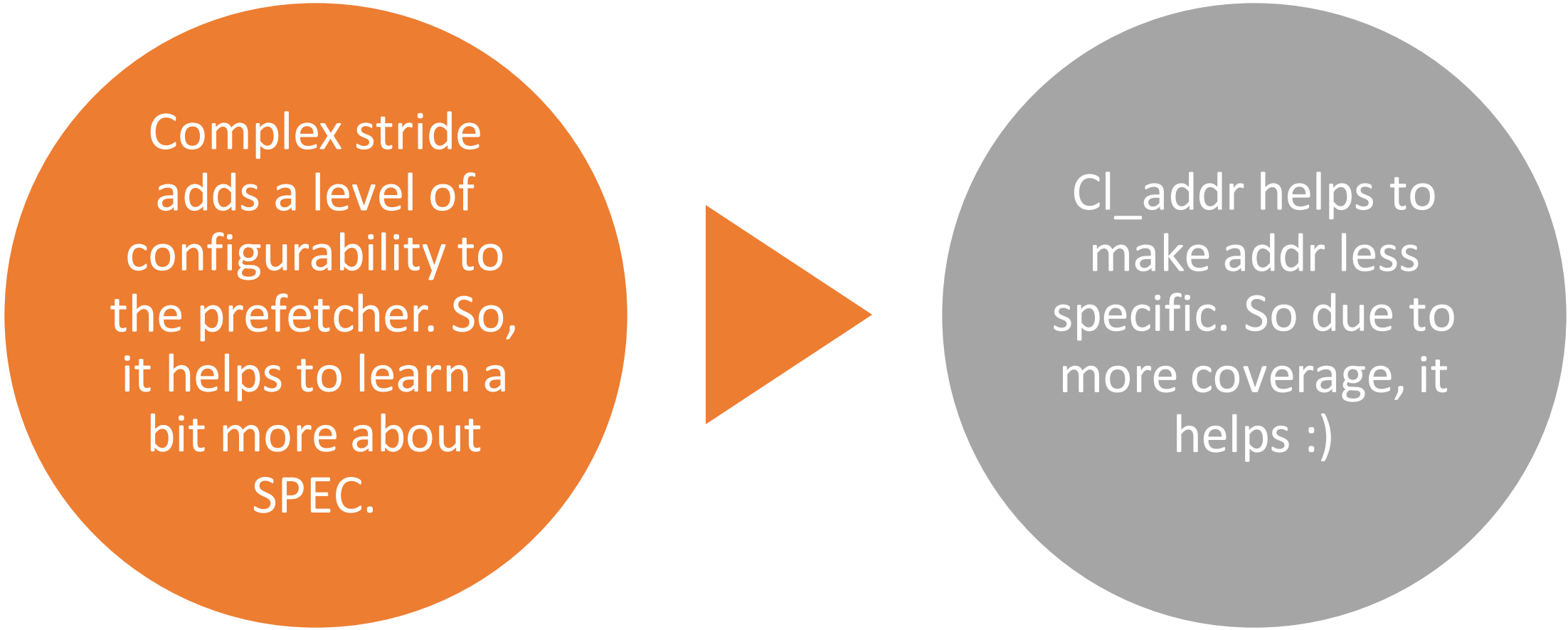
```
else if (trackers[cpu][index].pref_type == 0x300)
{
    int pref_offset = 0;
    for (int i = 0; i < prefetch_degree; i++)
    {
        pref_offset += trackers[cpu][index].stride;
        uint64_t pf_address = ((cl_addr + pref_offset) << LOG2_BLOCK_SIZE);

        I
        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

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

    uint64_t block_number = addr >> LOG2_BLOCK_SIZE;

    /* check prefetch hit */
    bool prefetch_hit = false;
    if (cache_hit == 1)
    {
        uint32_t set = get_set(block_number);
        uint32_t way = get_way(block_number, set);
        if (block[set][way].prefetch == 1)
            prefetch_hit = true;
    }

    /* check trigger access */
    bool trigger_access = false;
    if (cache_hit == 0 || prefetch_hit)
        trigger_access = true;

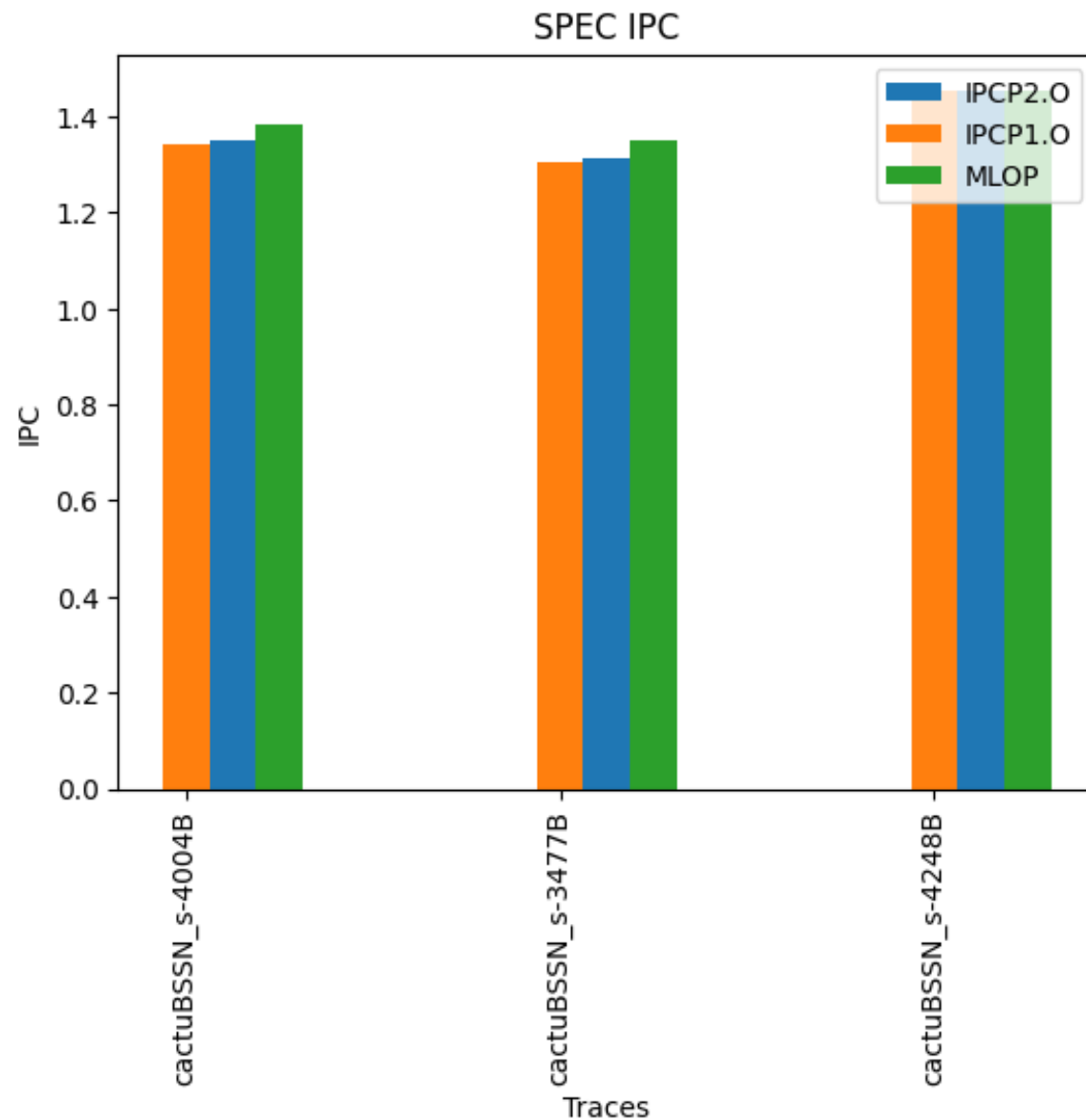
    if (trigger_access)
        /* update MLOP with most recent trigger access */
        L1D_PREF::prefetchers[cpu].access(block_number);

    /* issue prefetches */
    L1D_PREF::prefetchers[cpu].mark(block_number, L1D_PREF::State::ACCESS);
    num_prefs += L1D_PREF::prefetchers[cpu].prefetch(ip, addr, metadata, this, block_number);

    if (L1D_PREF::DEBUG_LEVEL >= 1)
    {
        L1D_PREF::prefetchers[cpu].log();
        cerr << "*****" << dec << endl;
    }

    /* stats */
    L1D_PREF::prefetchers[cpu].track(block_number);
}
```

# Integrating MLOP? Minor improvement



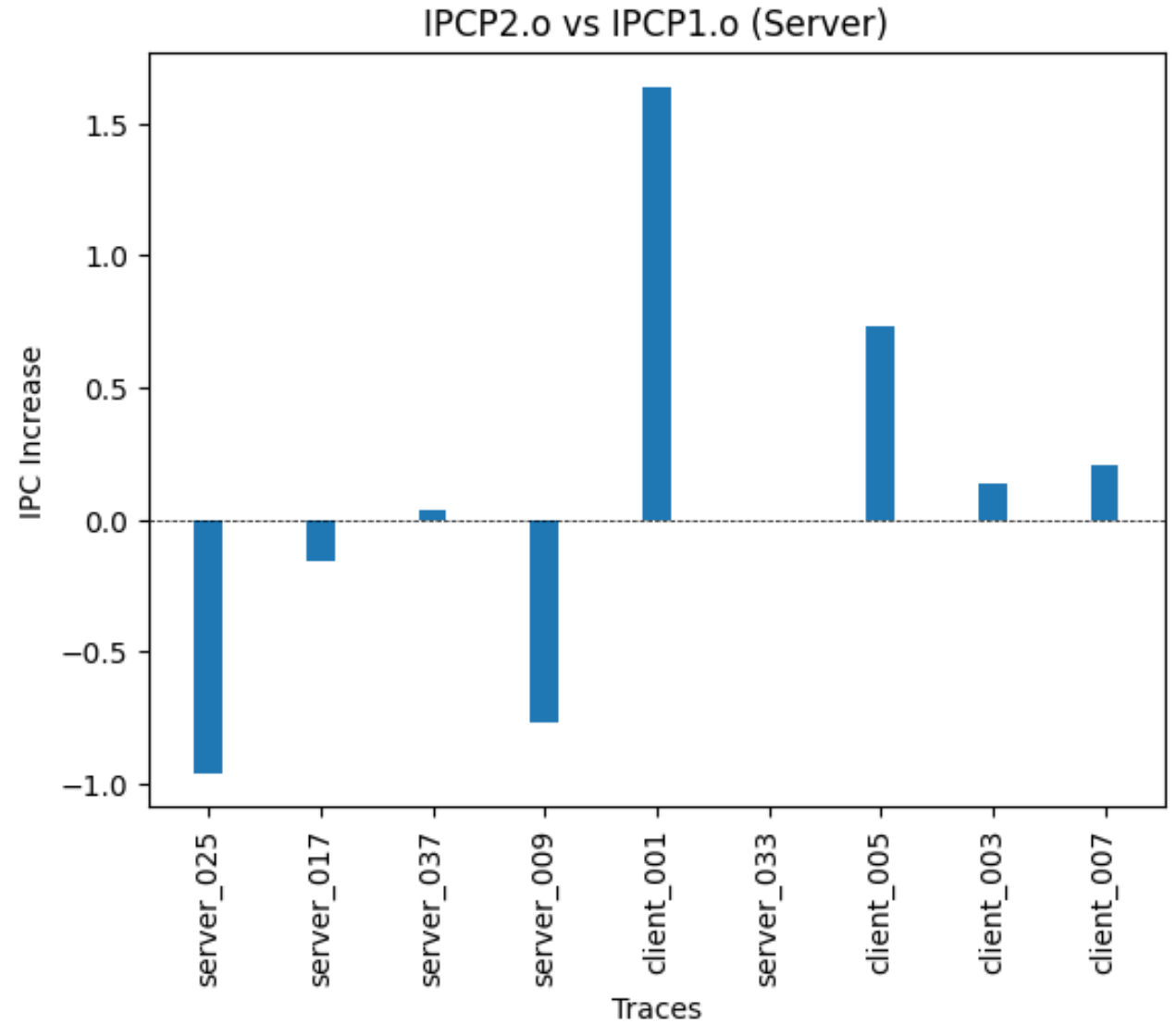


MLOP prefetcher tends to do well on cactus traces

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





Results!

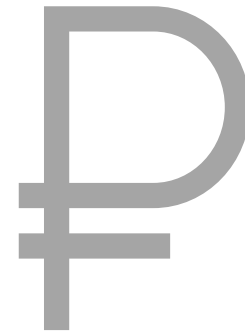
---

Traces	Speed-up
GRAPH	1.072
SAT	1.007
SPEC	0.999
SERVER	1.001

# Further improvements?



Perhaps scope for further optimization?



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

# Bouquet of DPC3 Winners

Ayush Agarwal\*, Sankalan Baidya<sup>†</sup> and Soham Joshi<sup>‡</sup>

Department of Computer Science, IIT Bombay

Mumbai, Maharashtra, India

Email: \* ayushagarwal@cse.iitb.ac.in, <sup>†</sup> somekoln@cse.iitb.ac.in, <sup>‡</sup> sohamjoshi@cse.iitb.ac.in

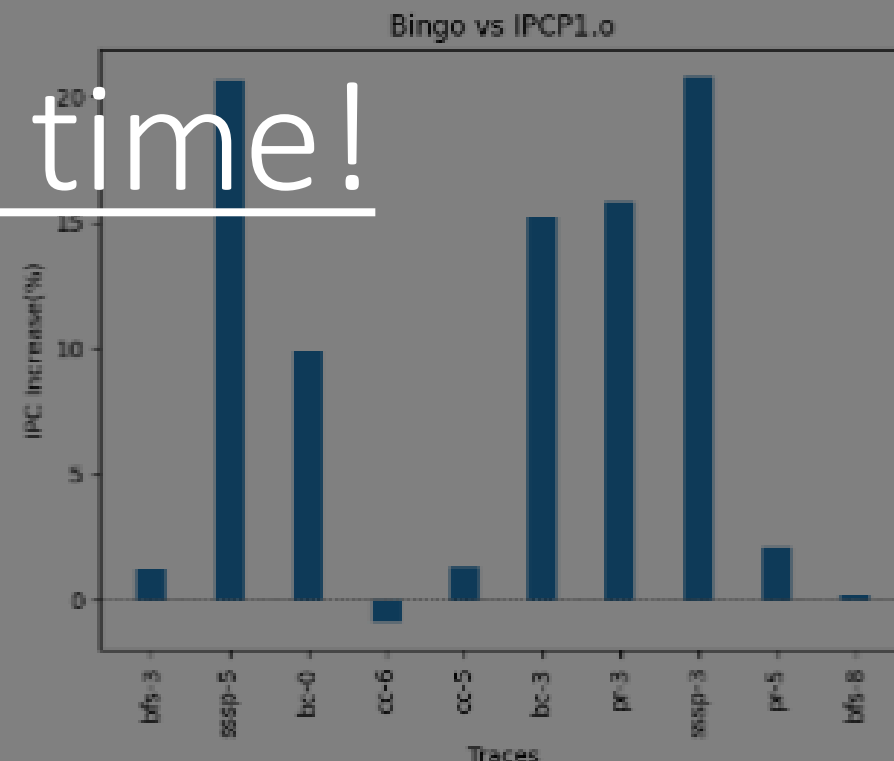
**Abstract**—This work is built upon IPCP 1.0 by Biswa, which uses a bouquet of prefetchers. We are proposing a series of specialized improvements for different categories like Graphs, Sat-solvers, SPEC traces and Server workloads, leading to a prefetcher which performs at par or better than IPCP for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

**Index Terms**—Prefetchers, 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.

Paper time!





The background of the image is a dense, repeating pattern of dark red roses. The roses are in various stages of bloom, with their petals tightly curled and layered, creating a rich, textured appearance. The color is a deep, slightly dark red, and the lighting is soft, highlighting the contours of the petals.

ଆପଣଙ୍କର ଦିନ ମଙ୍ଗଳମୟ ହେଉ