Instruction Prefetching Algorithms Comparison on PIN

(Amanda) Yilan Zhu
Northeastern University
EECE 7352 Computer Architecture

Introduction

- Instruction cache misses are crucial when evaluating the performance of microprocessors.
- Different strategies: Increase cache size/associativity; adding a victim cache/branch prediction…
- <u>Cache prefetching</u>: 2 tasks:
  - To "guess" which line will be soon accessed
  - Initiate the prefetch "just in time"
- If done correctly, prefetching could remove all cache misses.

Project Goals

- Simulate 3 cache prefetching algorithms:
  - Next-line prefetching
  - Target-line prefetching
  - Wrong-path prefetching
- Benchmark
  - Barnes.c - Simulates the gravitational forces acting on a star cluster using the Barnes-Hut n-body algorithm.
  - FindPi.c – Find the value of Pi by the Monte-Carlo method.
- Different Cache Set Associativity
  - Direct mapping/ 2-way/ 4-way

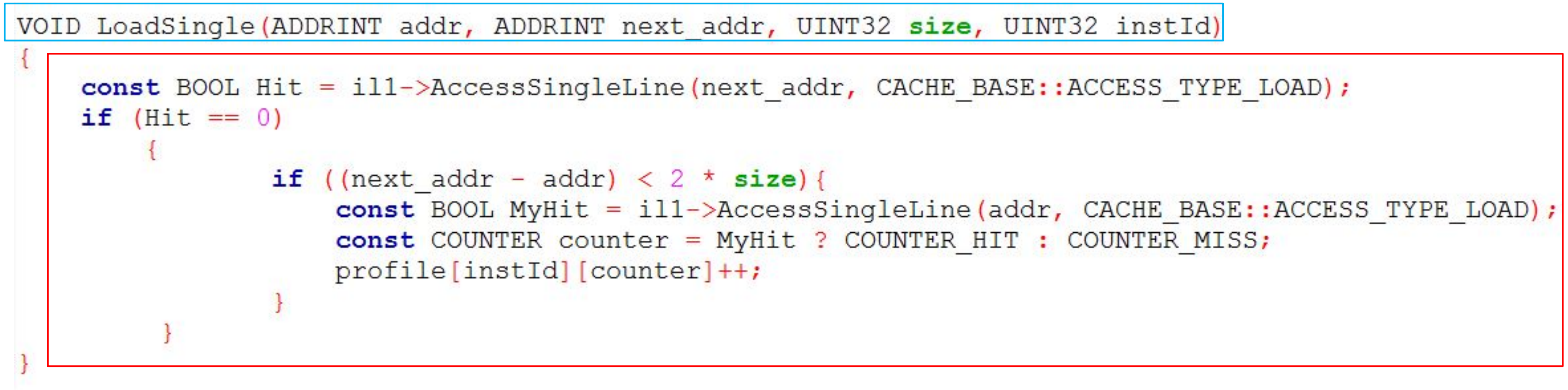
Tools

- Intel Pin 3.0
  - Tool for the “dynamic instrumentation of programs”,
  - Customize you own pintool to“simulate an instruction cache.”
  - Ran on COE nano (x86_64)
- Pintool : icache
  - Instruction cache simulator (icache.cpp)
  - Available in /pin/sources/tools/Memory
  - Modified to have prefetching abilities
  - Modifications made in both instrumentation and analysis routines

Prefetching Algorithms - 1

- Next-line prefetching
  - Prefetch next line if not in cache and when current line is in certain distance of next line.

In Analysis routine: executed at insertion points

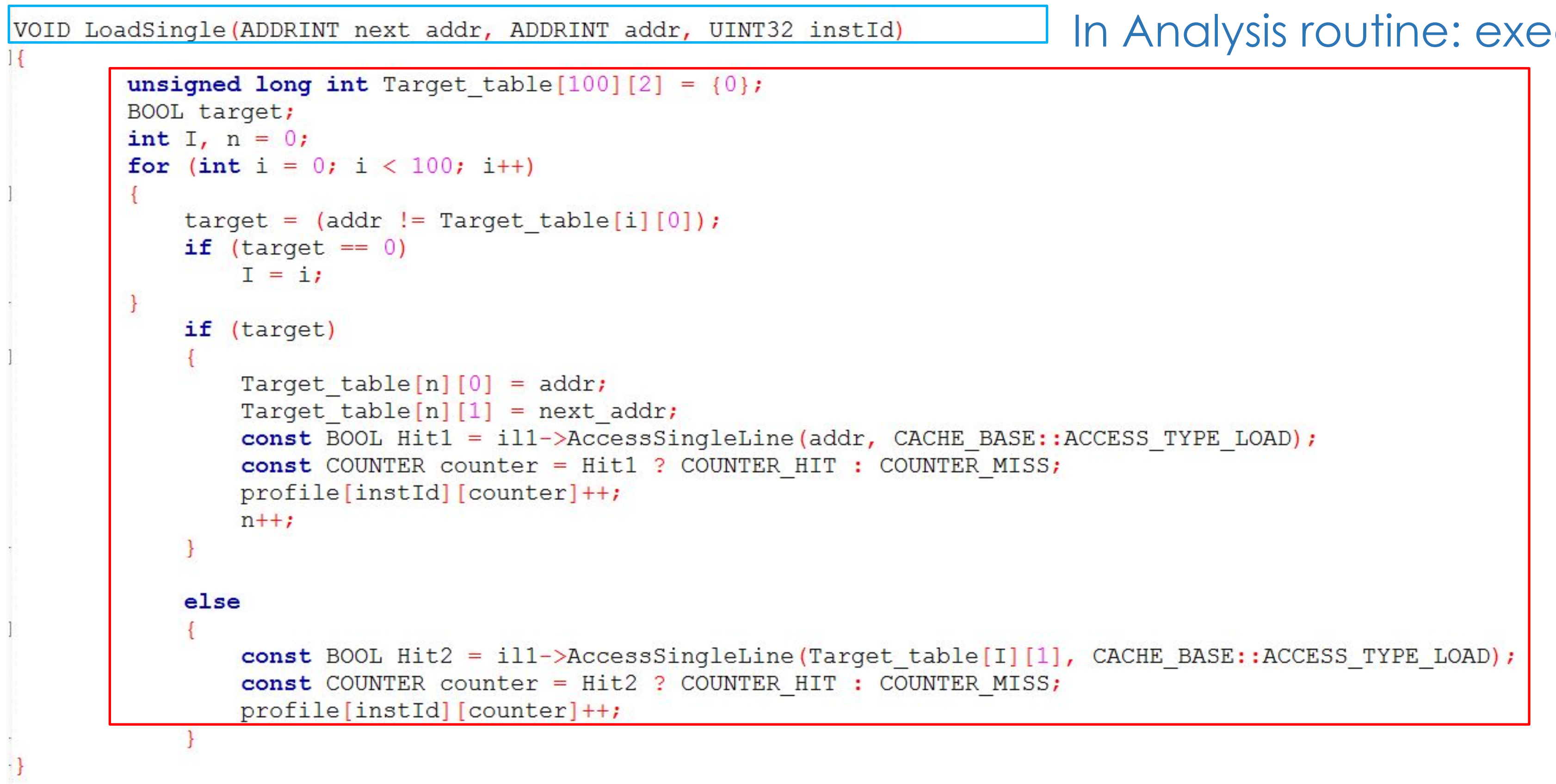


```
VOID LoadSingle(ADDRINT addr, ADDRINT next_addr, UINT32 size, UINT32 instId)
{
    const BOOL Hit = il1->AccessSingleLine(next_addr, CACHE_BASE::ACCESS_TYPE_LOAD);
    if (Hit == 0)
        {
                if ((next_addr - addr) < 2 * size){
                    const BOOL MyHit = il1->AccessSingleLine(addr, CACHE_BASE::ACCESS_TYPE_LOAD);
                    const COUNTER counter = MyHit ? COUNTER_HIT : COUNTER_MISS;
                    profile[instId][counter]++;
                }
          }
}
```

When the next line is not in cache (miss), it will be prefetched when the distance between the current and next address is smaller than 2 cache line sizes.

Prefetching Algorithms - 2

- Target-Line prefetching
  - Uses a maintained target table (current line – successor)
  - If successor line of current address in table is not in cache, prefetch.



```
VOID LoadSingle(ADDRINT next_addr, ADDRINT addr, UINT32 instId)
{
        unsigned long int Target_table[100][2] = {0};
        BOOL target;
        int I, n = 0;
        for (int i = 0; i < 100; i++)
        {
            target = (addr != Target_table[i][0]);
            if (target == 0)
                I = i;
        }
            if (target)
            {
                Target_table[n][0] = addr;
                Target_table[n][1] = next_addr;
                const BOOL Hit1 = il1->AccessSingleLine(addr, CACHE_BASE::ACCESS_TYPE_LOAD);
                const COUNTER counter = Hit1 ? COUNTER_HIT : COUNTER_MISS;
                profile[instId][counter]++;
                n++;
            }

            else
            {
                const BOOL Hit2 = il1->AccessSingleLine(Target_table[I][1], CACHE_BASE::ACCESS_TYPE_LOAD);
                const COUNTER counter = Hit2 ? COUNTER_HIT : COUNTER_MISS;
                profile[instId][counter]++;
            }
}
```

In Analysis routine: executed at insertion points

Creates a fixed size target table with current address and successor line.

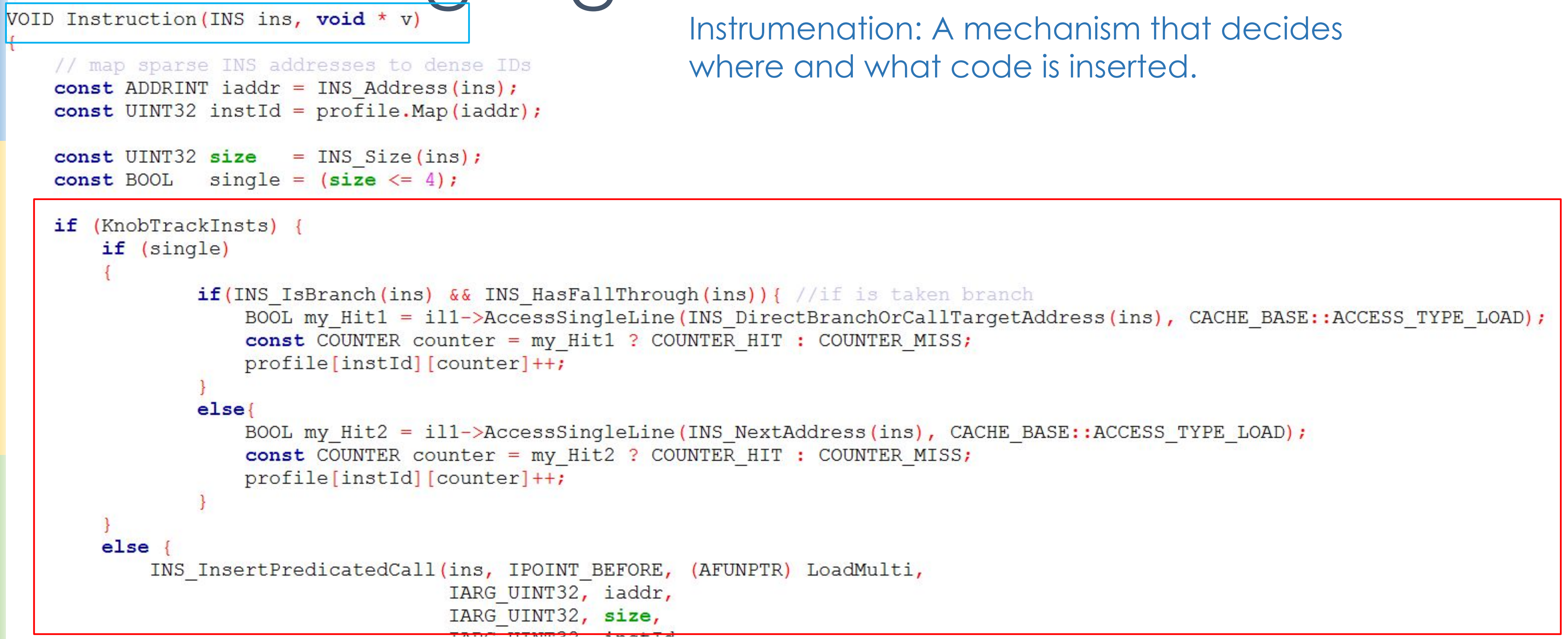
This will be passed to the instrumentation analysis function.

Prefetching Algorithms - 3

- Wrong-path prefetching
    - Combination of next-line and target prefetching.
        - Not-taken branch -> prefetch cache line with fallthrough address
        - Taken branch -> prefetch branch target line
    - Perform better than correct-path only schemes when there exists a large disparity between the CPU cycle time and the memory speed.
        - This is because other algorithms try to prefetch down target paths which will be executed almost immediately, and if memory is slow the prefetch may not be initiated soon enough.

Prefetching Algorithms - 3

Instrumenation: A mechanism that decides where and what code is inserted.



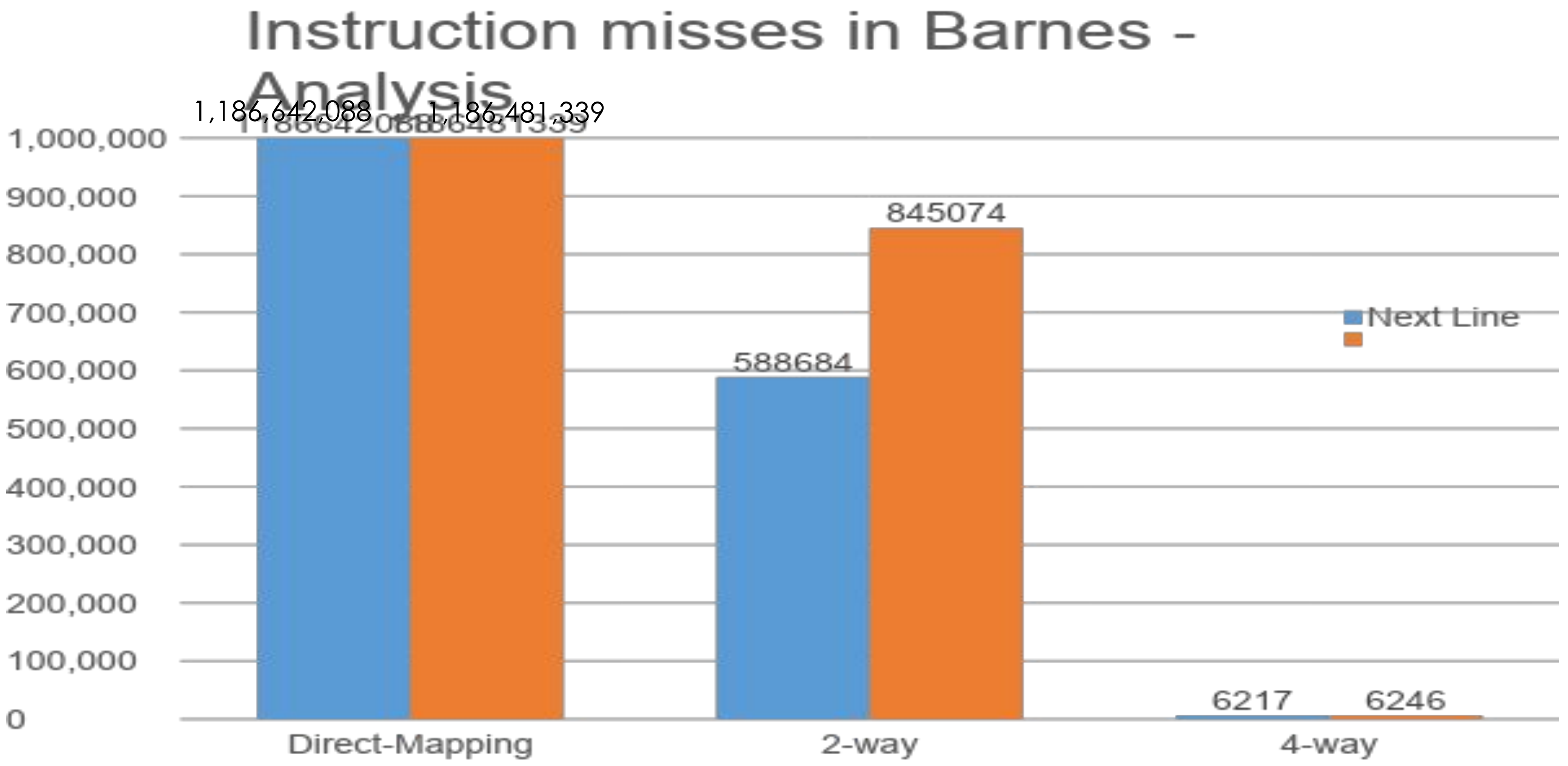
```
VOID Instruction(INS ins, void * v)
{
    // map sparse INS addresses to dense IDs
    const ADDRINT iaddr = INS_Address(ins);
    const UINT32 instId = profile.Map(iaddr);

    const UINT32 size   = INS_Size(ins);
    const BOOL   single = (size <= 4);

    if (KnobTrackInsts) {
        if (single)
        {
                if(INS_IsBranch(ins) && INS_HasFallThrough(ins)){ //if is taken branch
                    BOOL my_Hit1 = il1->AccessSingleLine(INS_DirectBranchOrCallTargetAddress(ins), CACHE_BASE::ACCESS_TYPE_LOAD);
                    const COUNTER counter = my_Hit1 ? COUNTER_HIT : COUNTER_MISS;
                    profile[instId][counter]++;
                }
                else{
                    BOOL my_Hit2 = il1->AccessSingleLine(INS_NextAddress(ins), CACHE_BASE::ACCESS_TYPE_LOAD);
                    const COUNTER counter = my_Hit2 ? COUNTER_HIT : COUNTER_MISS;
                    profile[instId][counter]++;
                }
        }
        else {
            INS_InsertPredicatedCall(ins, IPOINT_BEFORE, (AFUNPTR) LoadMulti,
                                     IARG_UINT32, iaddr,
                                     IARG_UINT32, size,
```

If is taken branch, fetch the target address;
if not taken branch, fetch next “natural” address

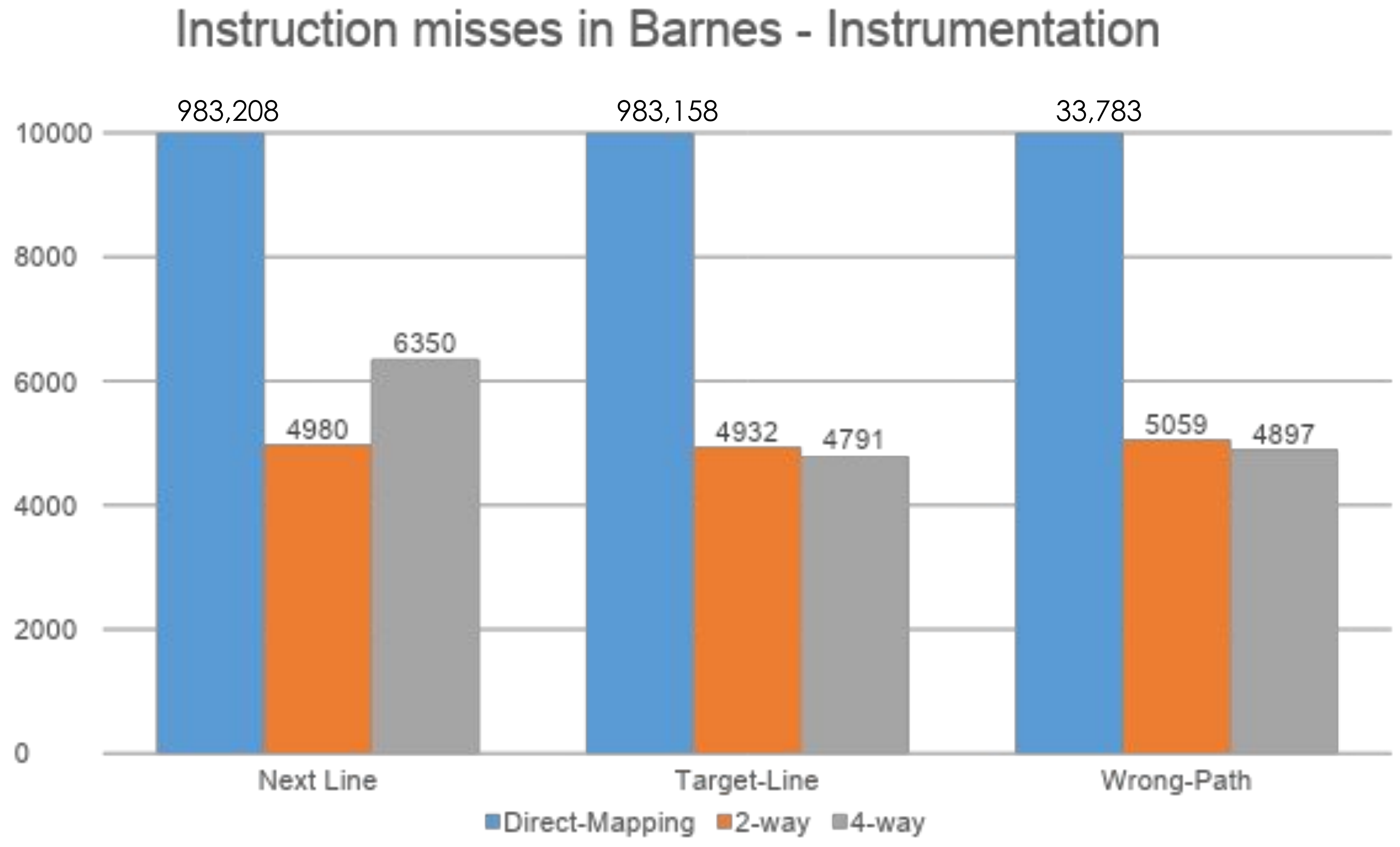
Experiment Design

- Next-Line prefetching and Target-line prefetching done on both analysis and instrumentation routine.
- Wrong-path only done on instrumentation.
- Direct-mapping/ 2-way / 4-way set associative.
- Benchmarks: FindPi.c / Barnes

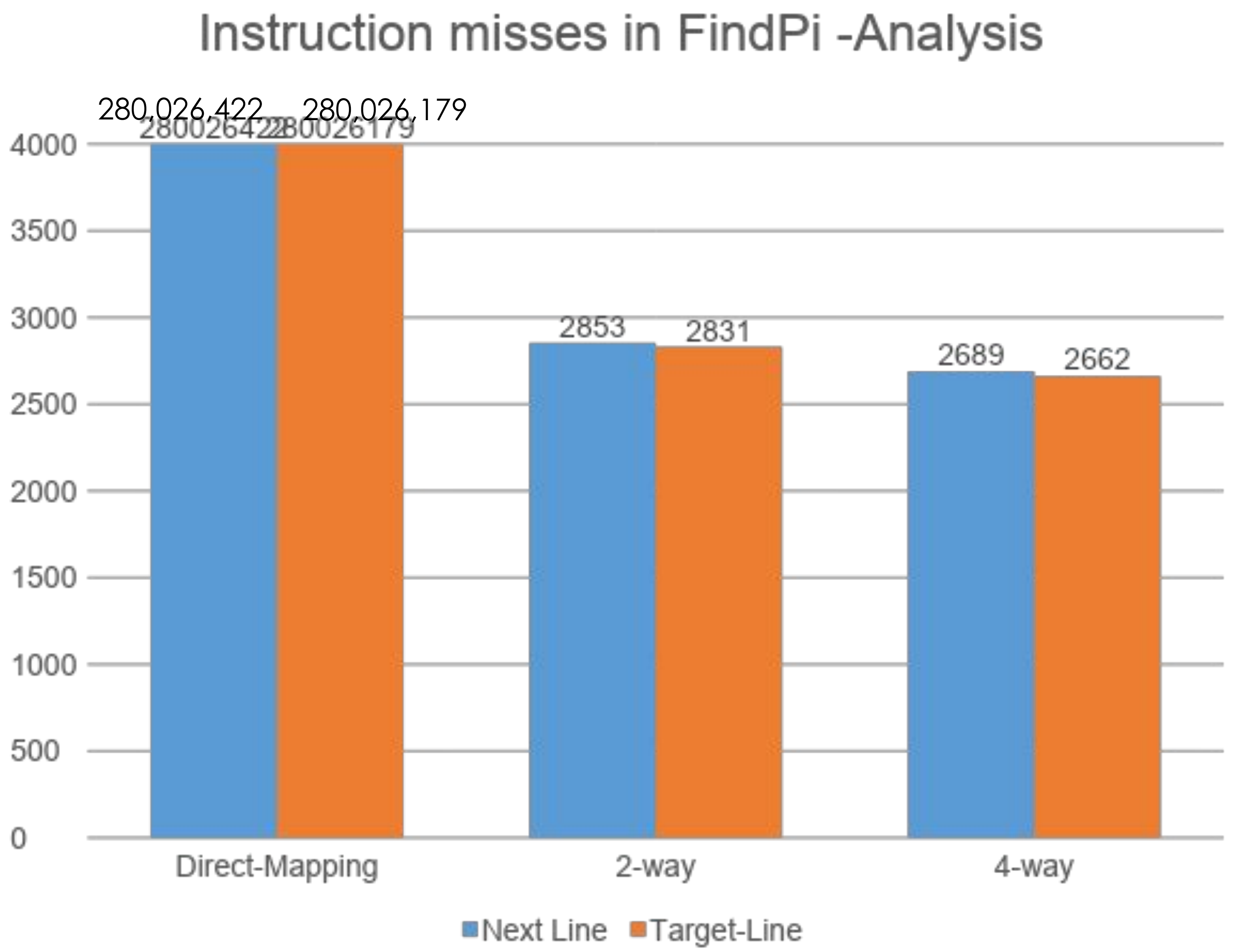
Results


Instruction misses in Barnes - Analysis
1,186,642,088
1,186,481,339
1186642088
1186481339
1,000,000
900,000
800,000
700,000
600,000
500,000
400,000
300,000
200,000
100,000
0
845074
588684
6217
6246
Direct-Mapping
2-way
4-way
🟦 Next Line
🟧

Results

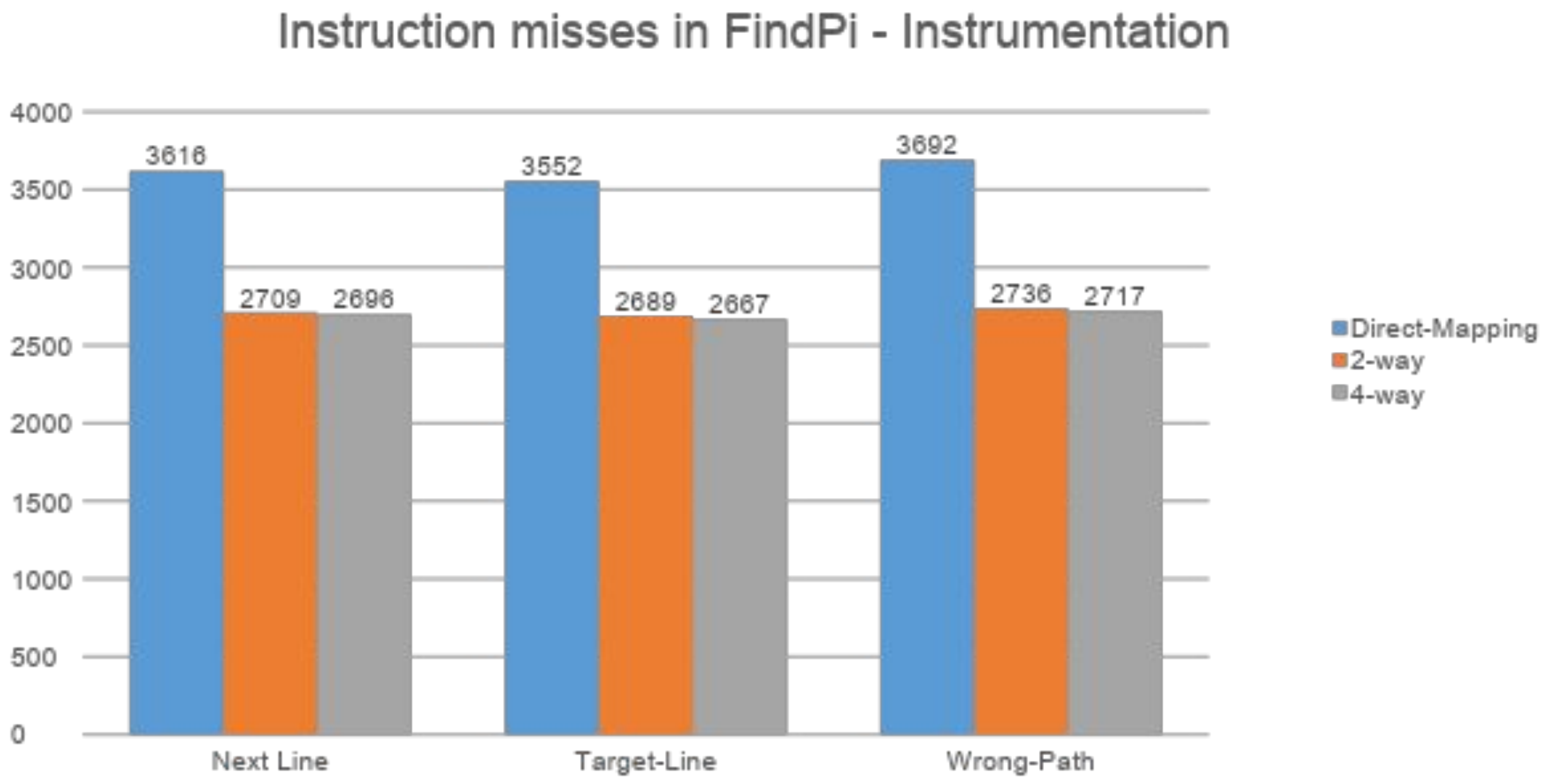

Instruction misses in Barnes - Instrumentation
983,208
983,158
33,783
10000
8000
6000
4000
2000
0
6350
4980
4932
4791
5059
4897
Next Line
Target-Line
Wrong-Path
Direct-Mapping
2-way
4-way

Results



Instruction misses in FindPi -Analysis
280,026,422
280,026,179
280026422
280026179
4000
3500
3000
2500
2000
1500
1000
500
0
2853
2831
2689
2662
Direct-Mapping
2-way
4-way
Next Line
Target-Line

Results


Instruction misses in FindPi - Instrumentation
4000
3500
3000
2500
2000
1500
1000
500
0
3616
2709
2696
3552
2689
2667
3692
2736
2717
Next Line
Target-Line
Wrong-Path
Direct-Mapping
2-way
4-way

Conclusions

- Set-associative improve cache hits significantly.
- Wrong Path prefetching, slightly improves cache hits
- Pintool modification should be done in Analysis.