30th October 2024

TEXAS
The University of Texas at Austin

Increasing Performance with B1 Bypass Instructions

Harshit Roy
PhD Student, The University of Texas at Austin

Regular Memory Access / Stride

Column in matrix

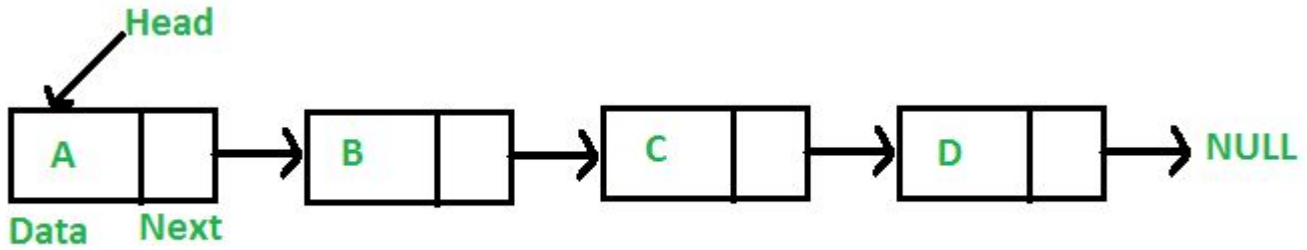
Elements in array of `structs`

- Access patterns often follow a *stride*
  - Example 1: Accessing column of elements in a matrix
  - Example 2: Accessing elements in array of `structs`
- Detect stride S, prefetch depth N
  - Prefetch X+S, X+2S, …, X+NS

Not the Target Application

Hardware Prefetcher - works

LinkedList Traversal

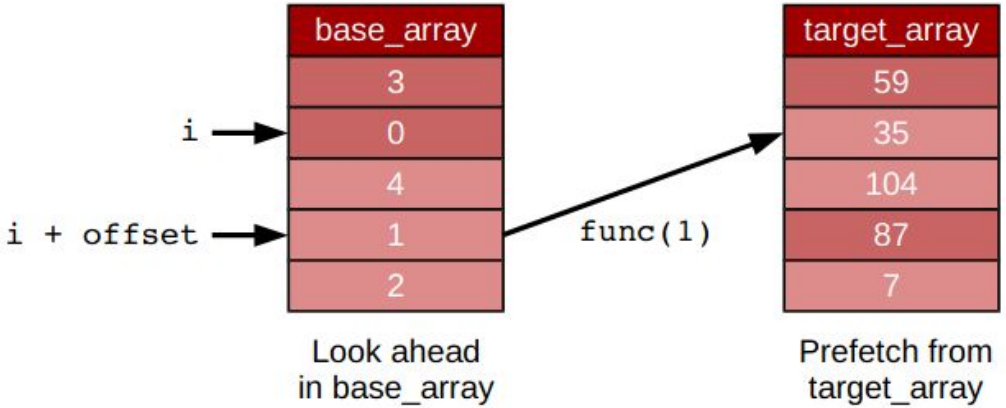



```
void traverse(Node* head) {
        Node* current = head;
        while (current != nullptr) {
                // Process current node
                process(current->data);
                // Move to the next node
                current = current->next;
        }
}
```

Not the Target Application
Can't Prefetch

Irregular Memory Access

```
1 for (i=0; i<base_array_size; i++) {
2   target_array[func(base_array[i])]++;
3 }
```



Target Application

Hardware Prefetchers → Don't work

S. Ainsworth and T. M. Jones, "Software prefetching for indirect memory accesses," *2017*

Software Prefetching

```
1 for (i=0; i<base_array_size; i++) {
2   target_array[func(base_array[i])]++;
3 }
```

```
1 for (i=0; i<NUM_KEYS; i++) {
2   // The intuitive case, but also
3   // required for optimal performance.
4   SWPF(key_buff1[key_buff2[i + offset]]);
5   // Required for optimal performance.
6   SWPF(key_buff2[i + offset*2]);
7   key_buff1[key_buff2[i]]++;
8 }
```

Brings key_buff1 and key_buff2 into the caches before the increment operation is executed → Reduces Latency Better Perf. ??

Camel Microbench

```
1 #define SIZE_OF_DATA 33554432
2 #define c_0 64
3 #define c_1 32
4
5 for (i=0; i<SIZE_OF_DATA; i++) {
6   SWPF(array[i+c_0]);
7   SWPF(*array_0[i+c_1]);
8   sum += hash(hash(hash...(*array_0[i])));
9 }
```

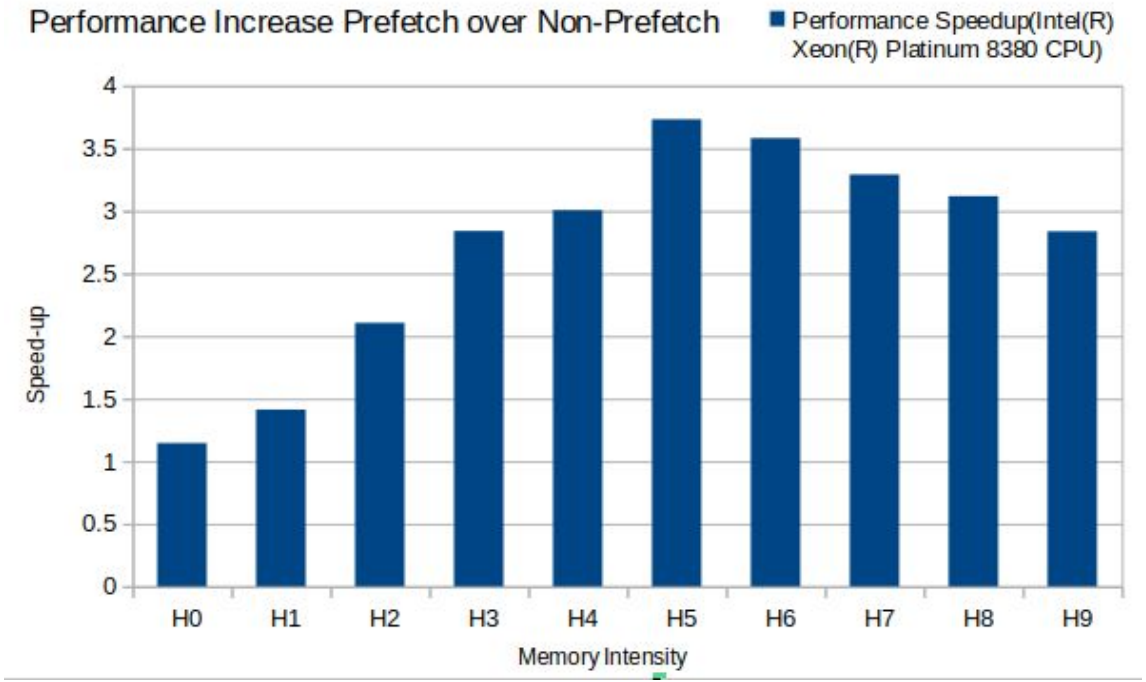
Hash → Compute

Increasing Hash Count → Decreases Memory Intensity

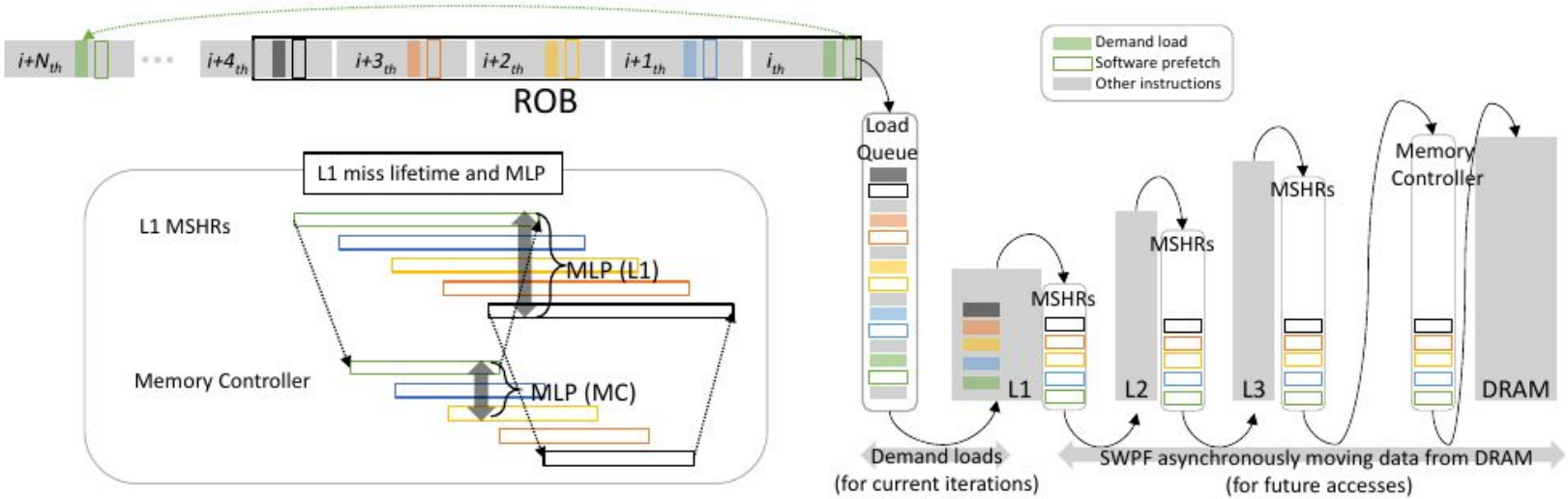
H0 → No Hashing (Most Memory Intensive)

H0 > H1 > H2 > H3 > ….. (Memory Intensity Order)

Performance Speedup

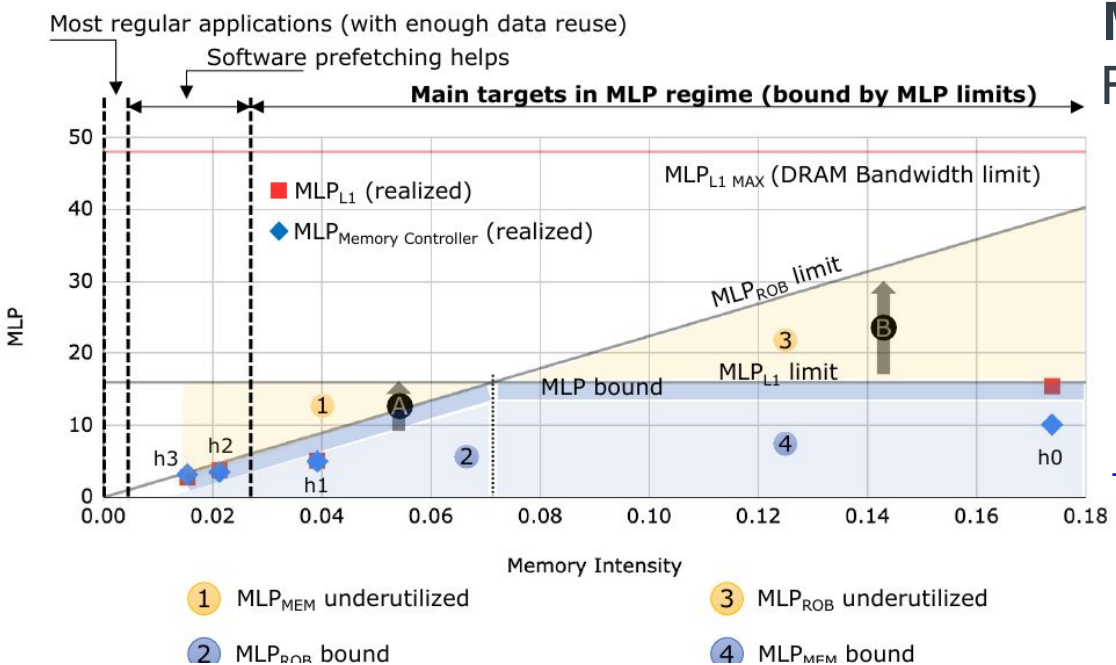


Memory Level Parallelism (MLP)



MLP → Memory Requests that are concurrently held by MSHRs at each cache level or scheduled in Memory Controller

MLP Roofline Model



MLP_max = Memory Pipeline Full or DRAM Bandwidth Limit

MLP_L1 = L1 MSHRs

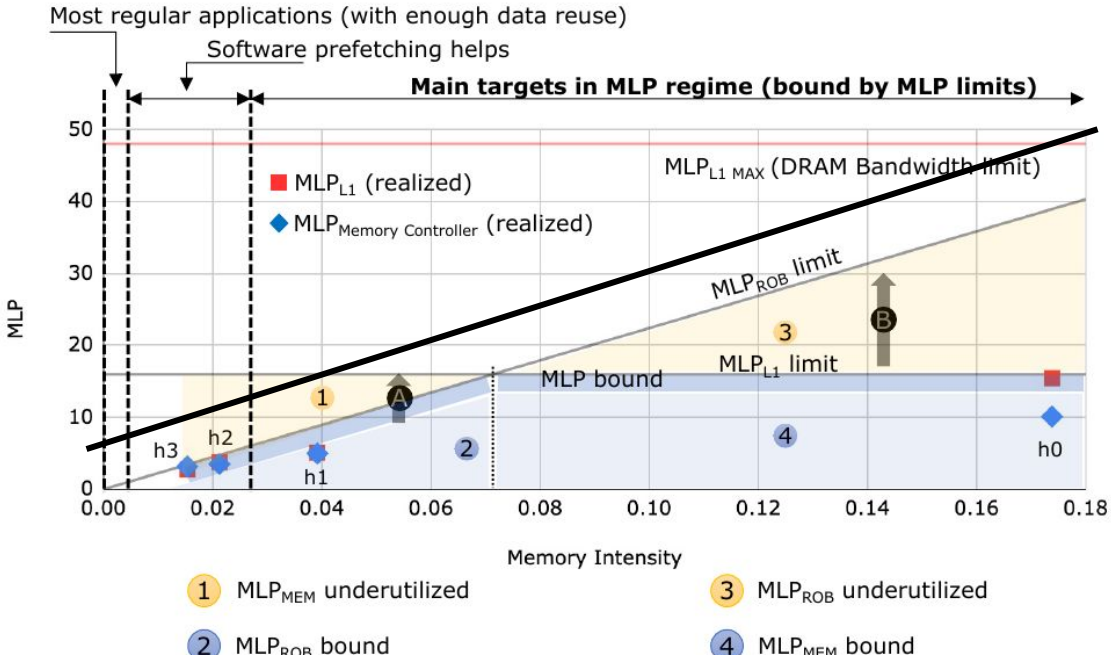
MLP_ROB = Increases with Memory Intensity

MLP_L1 < MLP_ROB

Increase L1 Mshrs

→expensive, will not work for h3, h2, ..

Effect of Software Prefetching (SWPF)

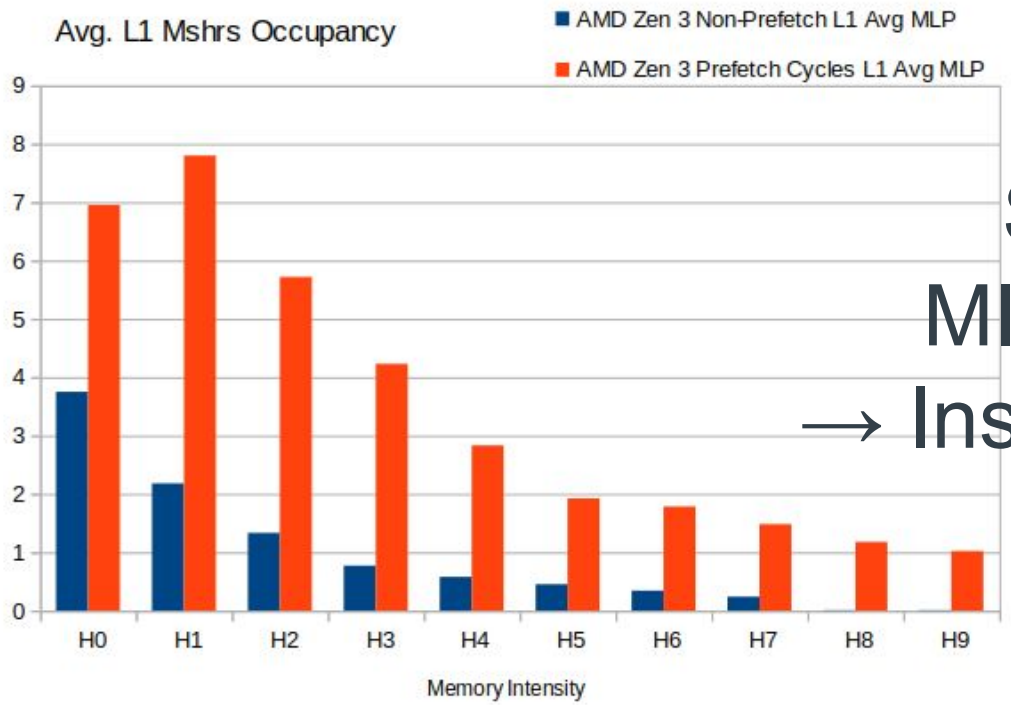


SWPF → Increases the ROB Limit

MLP h1, h2, … → Increases

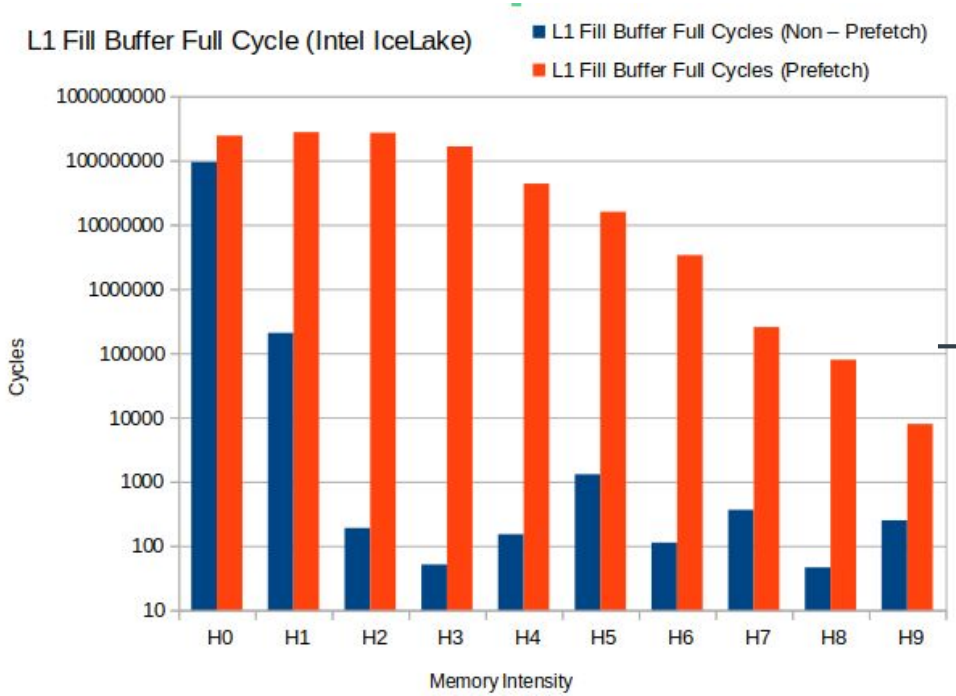
Remove L1 Mshrs limits??

Avg. L1 Mshrs Occupancy



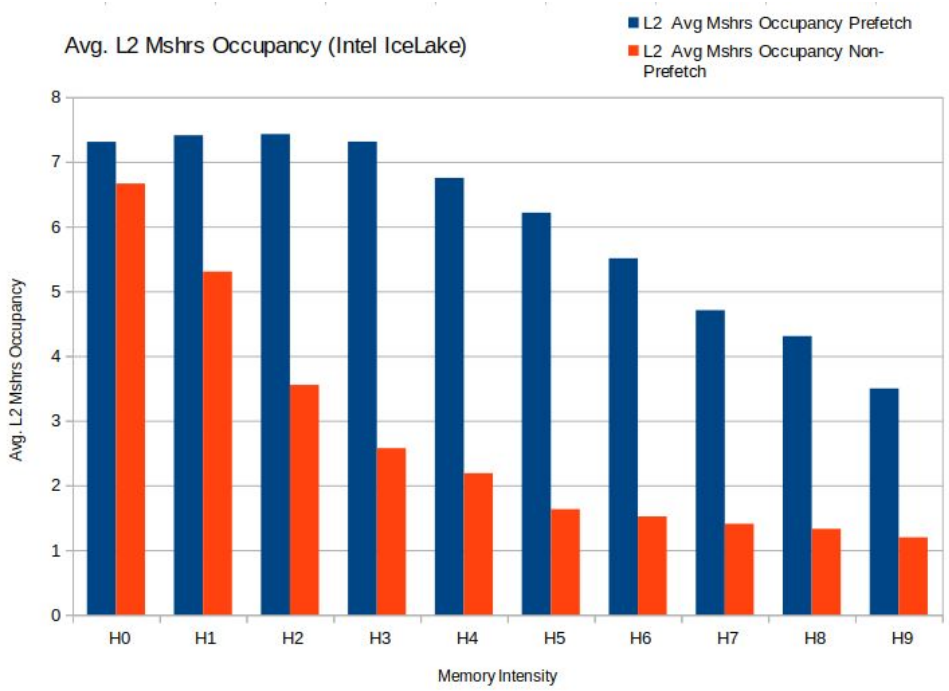
Saturation at
MI > H2 Prefetch
→ Insufficient L1 Mshrs

L1 Buffer Full Cycles

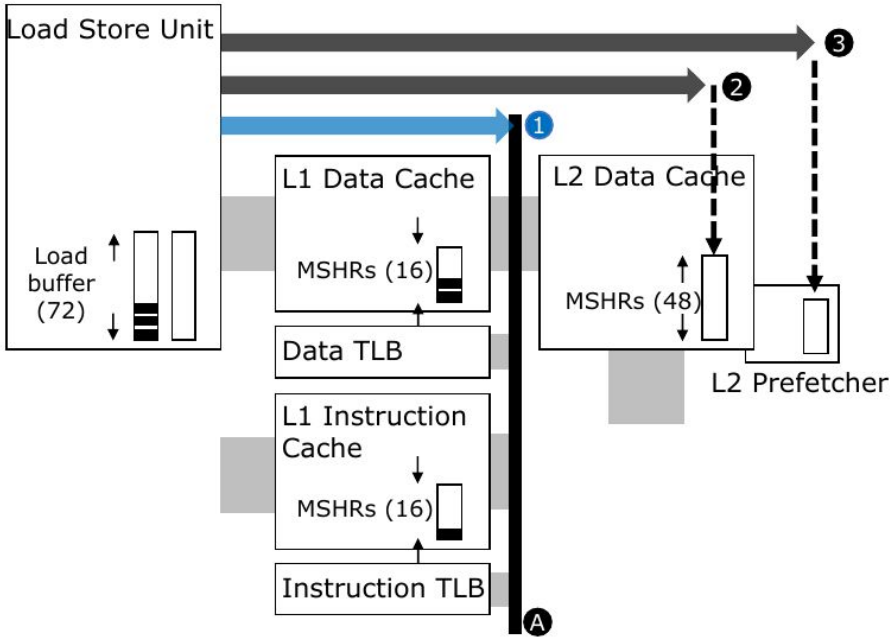


Saturation at
MI >= H2 Prefetch
→ Insufficient L1 Mshrs

Avg. L2 Mshrs Occupancy

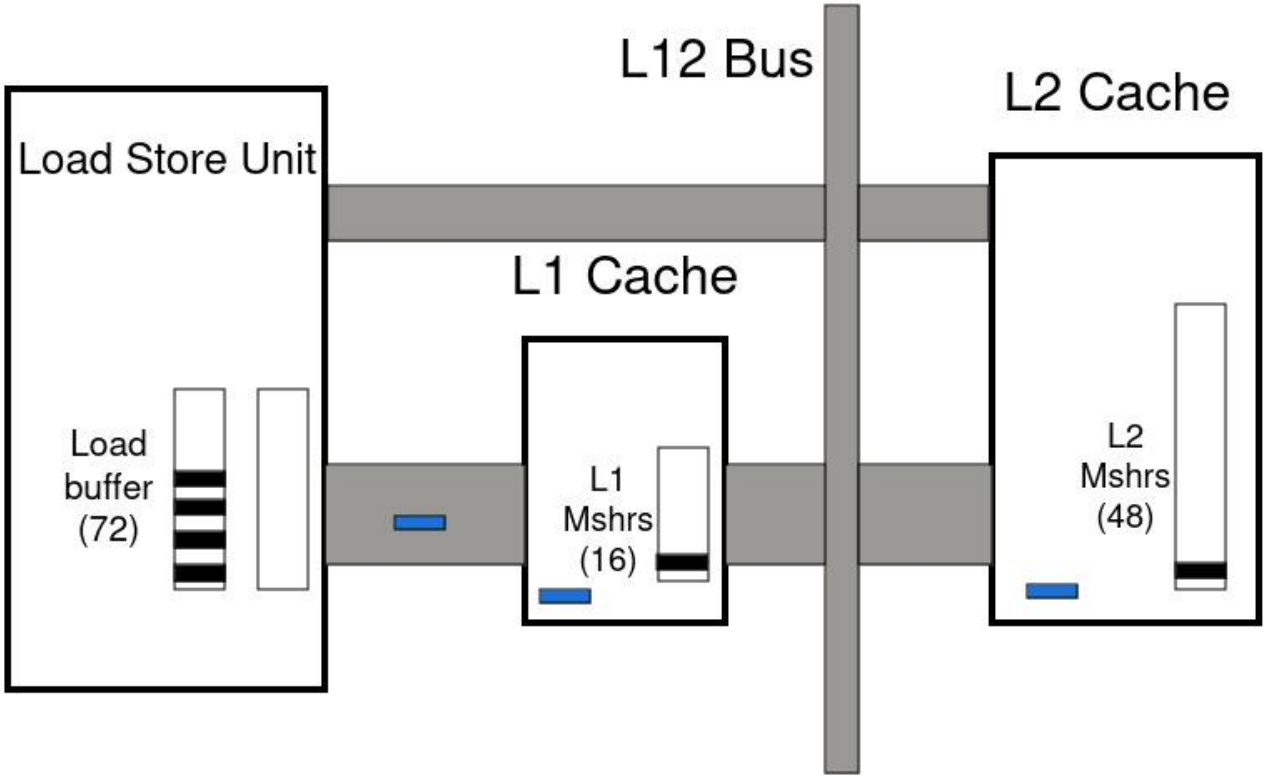


Bypass Prefetch Instruction (B1 SWPF)

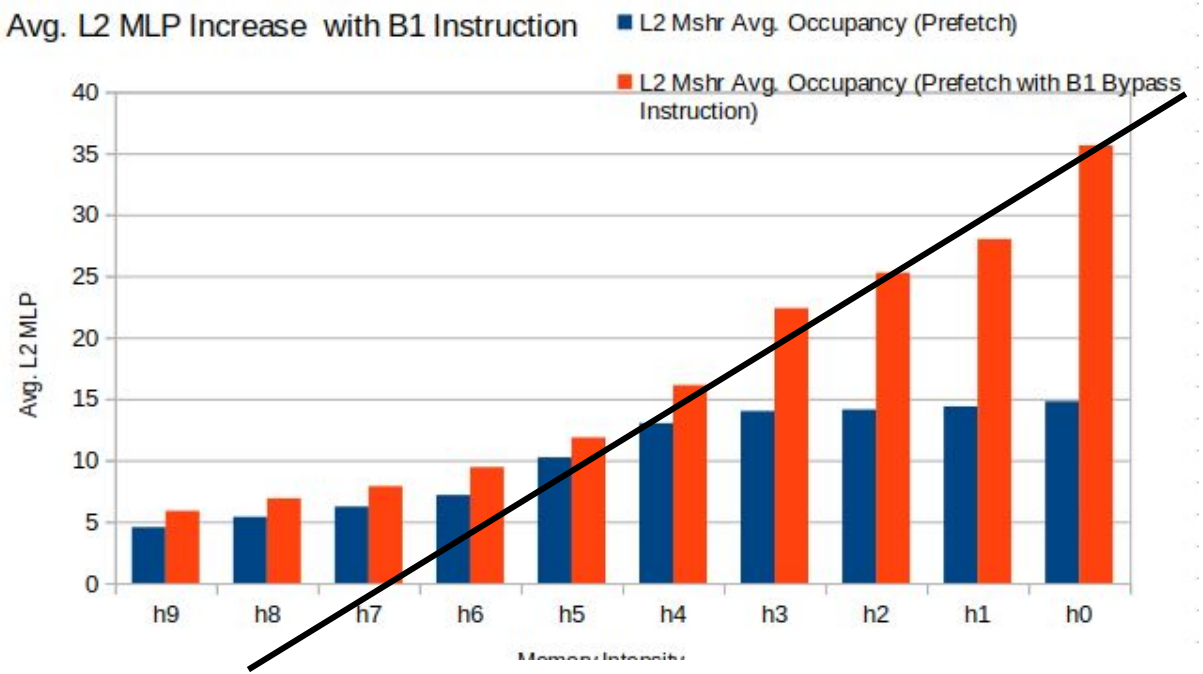


B1 SWPF: Directs Prefetch Requests to L2 MSHRs, Skipping L1 MSHRs. Extra Latency for Fetching Prefetched Data from L2 to L1 Upon Demand

B1 Bypass Software Prefetch



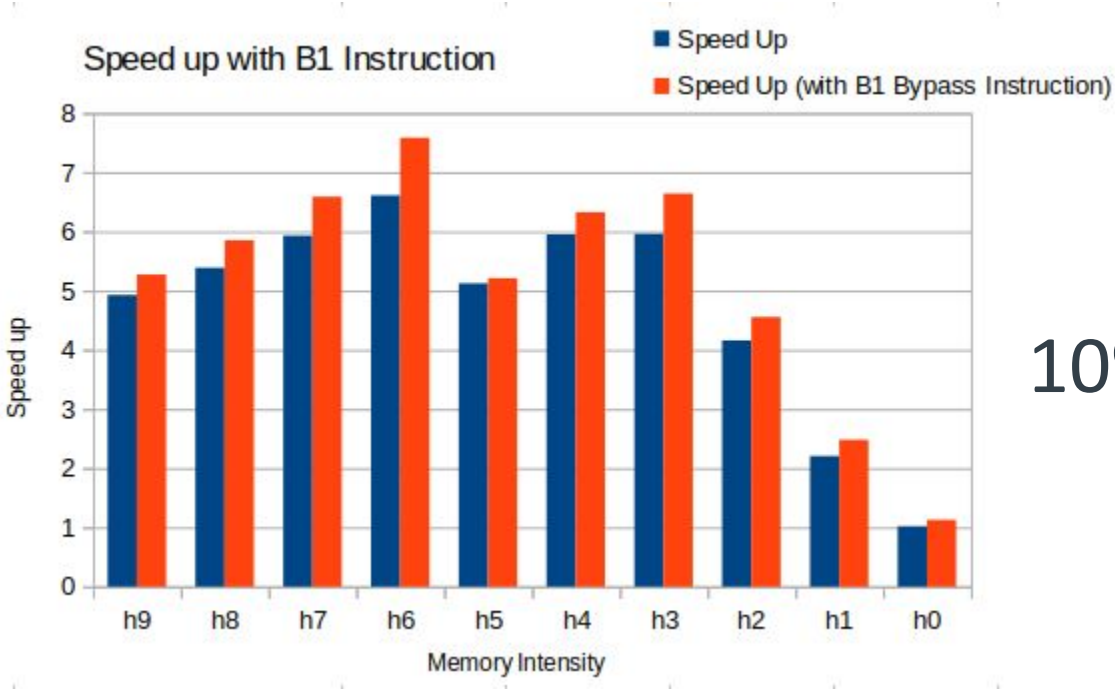
Evaluation (Gem5)



B1 SWPF ⟶ ROB Constraints only

SWPF ⟶ L1 MSHR Constraints

Performance Speedup (B1 SWPF)



10% Improvement

Thank You