CaRP : Cache Replacement Policies for Linked Data Structures

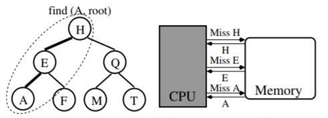
Tushar Goyal, Rini Patel, Aditi Sinha

Problem Statement

Accessing pointer-based linked data structures (LDS) such as linked lists, hash tables, B-trees exhibit highly irregular memory accesses.

Problem with pointer-chasing programs:

- Poor locality of data
- More cache misses
- Memory latency
- Pre-fetching is not helpful
- Limits parallelism

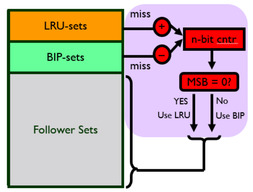


(a) Binary tree (b) Traditional architecture

Existing Policies

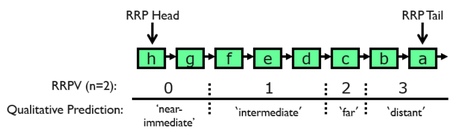
DIP:

- BIP: Insertion at LRU position and MRU with low probability (1/32)
- Set-Dueling between LRU and BIP
- Follower sets work based on pSel counter
- Works well for thrashing workloads, but not good for LDS.



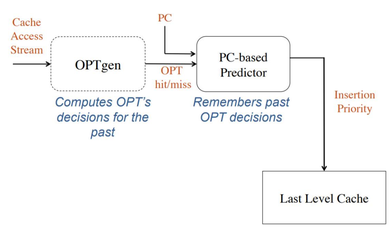
DRRIP:

- Tries to predict the future reference interval
- Inserts lines with RRPV, higher value means farther in future reference.
- Works well with scan and thrashing workload, but not with LDS.



Hawkeye:

- Train predictor per-PC based on past OPT behavior
- Predictor decides between cache friendly or cache averse data, and accordingly applies RRPV.



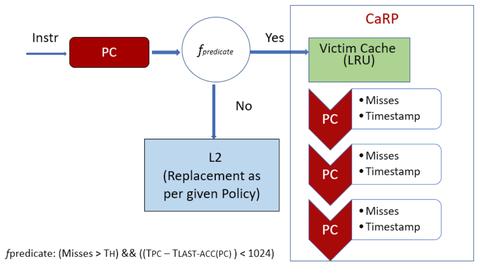
Proposed Policy

- Pointer Chasing workloads are in iterative or recursive forms.
- Typically small set of instructions with non-uniform memory access pattern.

```
/* Given a binary tree, print its nodes in inorder*/
void printInorder(struct node* node)
{
    if (node == NULL)
        return;

    /* first recur on left child */
    printInorder(node->left);

    /* then print the data of node */
    int val = node->data;

    /* now recur on right child */
```

```
printInorder(node*):
4006a655    push rbp
4006a748 89 e5    mov rbp,rsp
4006aa48 83 ec 20    sub rsp,0x20
4006ae48 89 7d e8    mov QWORD PTR [rbp-0x18],rdi
4006b248 83 7d e8 00    cmp QWORD PTR [rbp-0x18],0x0
4006b774 2b    je 4006e4 <printInorder(node*)+0x3e>
4006b948 8b 45 e8    mov rax,QWORD PTR [rbp-0x18]
4006bd48 8b 40 08    mov rax,QWORD PTR [rax+0x8]
4006c148 89 c7    mov rdi,rax
4006c4e8 dd ff ff ff    call 4006a6 <printInorder(node*)>
4006c948 8b 45 e8    mov rax,QWORD PTR [rbp-0x18]
4006c8b 00    mov eax,QWORD PTR [rax]
4006cf89 45 fc    mov DWORD PTR [rbp-0x4],eax
```

- Track last 1024 PCs for consecutive misses on that PC.
- Apply heuristic to select the Victim cache for PC if misses are above Th.
- 64 lines Victim Cache and Th = 4.

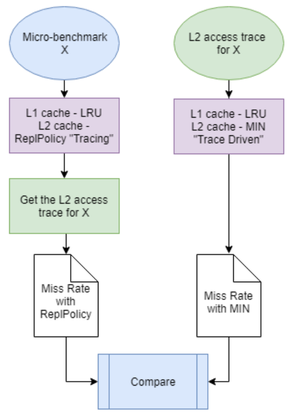


*f*predicate: (Misses > TH) && ((TPC – TLAST-ACC(PC)) < 1024)

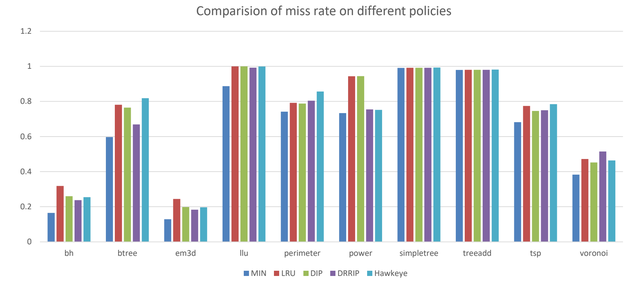
Experimental Evaluation

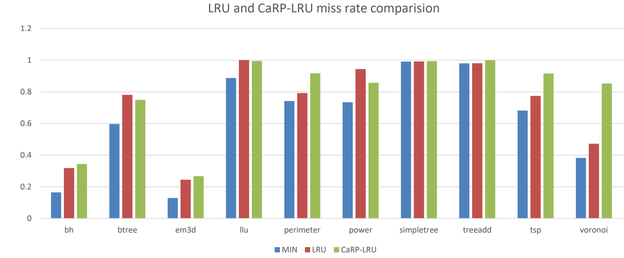
- Simulator : Zsim
- Machine : Ubuntu 12.04 with Pintool 2.14
- Memory hierarchy:
  - L1 - 32 kB (4-way, 5 cycle latency)
  - L2 - 256 kB (8-way, 12 cycle latency)
  - L3 - 2 MB (16-way , 35 cycle latency)
- Max simulation time = 720 sec

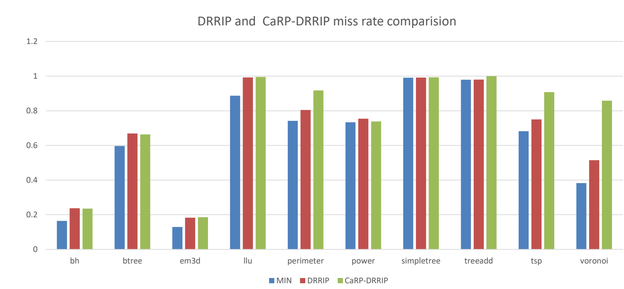
Benchmark	Data Organisation	Parameters
BH	Heterogenous OcTree	100000 bodies, 32 nodes
Em3D	Single Linked Lists	2000 H 500 E Nodes
Perimeter	Quad-Tree	11 levels
Power	N-way Tree, single-linekd Lists	N/A
TreeAdd	Binary-Tree	25 levels
Tsp	Balanced binary-tree	2 Million Nodes
Voronoi	Balanced binary-tree	1 Milllion Nodes
Simple Tree	Binary Tree	100 levels
B-Tree	B+ Tree	3 Million Nodes



Results







Conclusions

- Given workloads have little scope of improvement over MIN. (Max 8-10 %)
- CaRP with DRRIP achieves behavior equivalent to MIN on *power* benchmark.
- CaRP with LRU performs better than LRU on *power, llu*, and *btree*.

Future Work

- Select optimal Victim Cache size, hardware overhead
- Minimize the overhead of keeping PC history, buffering
- Try to incorporate memory accesses along with PC history to reduce the false positives