Lecture 19 (Instruction and Data Prefetching)

1 Instruction Prefetching

1.1 Next Line Prefetching

Spatial locality

1.2 Markov Prefetching

1. High correlation between consecutive misses
2. Given that X incurs a miss, predict the line which will incur a next miss
3. Can have n history as well

1.3 Call Graph Prefetching

1. Function call is predictable
2. Predict and prefetch the function

1.3.1 Hardware Approach Call Graph History Cache (CGHC)

1. Each entry contains a list of functions to be executed
2. Index is 1 initially
3. On returning from func1, prefetch func2

2 Data Prefetching

2.1 Stride Based Prefetching

1. Reference Prediction Table (RPT)
2. Each entry is made up of: instruction tag, previous address, stride, state

2.2 Extension

1. Decide when to prefetch
2. Needs to be dynamic

3. Depends on code in the loop

2.3 Pointer Chasing

1. No visible hardware pattern
2. Can insert code to actually prefetch node->next
3. `gcc_intrinsics.h` exists for this

exists a term called black belt programmer

2.4 Runahead Mode

1. Misses in L1 (especially L2) lead to stalls since IW and ROB fill up
2. Idea is to do some work during stall period

2.4.1 Implementation

1. Return a a junk value for the miss
2. Restart execution with junk value - add INV (invalid) bit
3. This is useful since we will prefetch data and train predictors
4. Once miss returns, flush instructions and re-execute instructions
5. For data requests during this time, maintain a runahead L1 cache

2.4.1.1 Runahead L1

1. If store is INV because of data (not because of address), prefetch this address
2. Can maintain additional state of INV and INV store

2.4.1.2 Loading Value

1. Try forwarding in LSQ
2. Else, access runahead cache
3. If miss, try accessing from L1
4. Else, load from L2 (prefetching)

2.5 Helper Threads

1. Spawn threads to execute backward slice of load
2. Backward slice determines the address of load
3. If resources available, saves time