Prefetching and Prefetchers

Andrew Lenharth
CS378 – fall 2014

Caches are (Mostly) Reactive

- Caching looks at past behavior because it correlates with future behavior
  - Temporal/Spatial locality
- Predicts future accesses will be the same or very near past accesses
- Use previous misses to prevent future misses

Wanted: proactive system

- Predict future misses before they happen
  - To locations not seen before
- Take action to prevent future misses
  - Confidence?

The core (predictable) access pattern

- Loops over arrays exist
  - Or pointers with fixed increment
  - Affine indexing ax+b

for (int x = 0; x < sizeof(A); ++x)
  ...A[b*x+c]...

(b,c are constant)

To predict the future

- Consider each instruction
- If addresses follow linear model (ax+b), then we can generate future addresses
  - Or any other pattern we want
- How far into the future can we predict?

MMM

for (int i = 0; i < n; ++i)
  for (int j = 0; j < n; ++j)
    for (int k = 0; k < n; ++k)
      C[i,j] += A[i,k] * B[k,j]

//Where
C[i,j] = (&*C)[i*n + j] //etc for A, B
Key Feature

- Future accesses can be computed by linear regression
- Many “streams” are in use
  - In MMM, A,B,C are separate streams
  - Need to compute streams separately
- Noisy
  - Lots of ops not related to stream accesses

Take action to change the future

- Load address from memory early
  - Into which cache?
  - How far into the future?
  - What if the load crashes the program?
  - How confident are we about the address?
- Two ways: hardware and software
  - Neither are ideal
  - You can do both
  - It’s hard, but possible to turn off the hardware

Hardware prefetchers

- A set of units which try to identify streaming accesses
  - Possibly at each cache level
- Latch onto a stream and predict future addresses
- Figure out how many cycles ahead to issue loads so data is ready when the code needs it
- Issues prefetch load
  - Not normal load

Why not load next?

1 ahead, 1 cycle latency  1 ahead, 3 cycle latency  3 ahead, 3 cycle latency

Why not do a load?

- Given: A[x] where A is size n
- What happens when loading A[n+1]?
  - Hardware doesn’t care
  - Page fault
    - OS might halt program with a segfault
  - Page fault
    - OS might load page which isn’t needed
Limitations

• Steals resources from real work
  – You loads and store are more important than possible future loads and stores
  – Multi-threaded interference
• Page limit – won’t prefetch into new page
• Limited to regular data accesses
  – Complexity of access pattern varies
  – How would you prefetch a binary tree?

Software prefetching

• Prefetch instruction
• Hard to write low overhead adaptive code
  – Adaptive to machine conditions
  – Adaptive to input sets
  – Adaptive to machine
• But can prefetch irregular accesses
  – But it is hard

MMM

```c
for (int i = 0; i < n; ++i)
    for (int j = 0; j < n; ++j)
        for (int k = 0; k < n; ++k) {
            C[i,j] += A[i,k] * B[k,j]
            if (k < n - delta)
                prefetch(A[i,k+delta], B[k+delta,j])
            if (k == n - delta)
                prefetch(C[i,j+1], B[0,j+1])
        }
```