Prefetching

- Try to keep instructions/data in cache
  - Increase cache size
  - Increase block size
  - Increase associativity
  - Victim caches
  - Others ...

- Caching isn’t perfect -> will always have misses
  - Capacity misses (especially data)
  - Compulsory and conflict misses
  - Coherence misses (caching can do nothing)

- Prefetching: start bringing missing instructions/data into cache early, so that they arrive on time.

- Caching: latency reduction; Prefetching: latency tolerance
Prefetching Issues

- What to prefetch?
  - Determine cache miss stream
  - Can be done by hardware or compiler

- When to prefetch?
  - Time prefetches so that instructions/data arrive when they are needed: Prefetch distance
  - Not too late, not too early

- Goal:
  - High coverage
  - Good timeliness
  - High accuracy
  - High usefulness
What to Prefetch: Memory Access Patterns

- Instructions
  - Often fit in L1. If not, usually fit in L2.
  - L1-L2 latency matters -> instruction prefetching can dramatically improve performance for large programs (e.g., databases)
  - Lots of spatial locality, except for taken branches

- Data
  - Very large working sets possible
  - Prefetch all the way from DRAM: large prefetch distance
  - Many data cache misses occur in loops:

    ```
    for (i = 0; i < N; i += S) {
        ... = A[i];
    }
    ```

    ```
    for (i = 0; i < N; i ++) {
        ... = A[B[i]];
    }
    ```

    ```
    ptr = root;
    while (ptr != NULL) {
        ptr = ptr->next;
    }
    ```

- Affine Array: Striding
- Indexed Array: Random
- Pointer Chasing: Random
Hardware Prefetching

- L1 Cache
- Predictor
- Prefetch Buffers
- L2 Cache
- Processor

Arrows indicate data flow:
- "L1 miss stream" from L1 to Predictor
- "Directs" from Predictor to Prefetch Buffers
- "Prefetch" from Prefetch Buffers to L2 Cache
- "Prefetch" from Processor to L1 Cache
Next-Line Predictor

- On a miss, fetch block and next N sequential blocks as well
  - Place missed block in cache
  - Place N prefetched blocks in prefetch buffer
- On processor access, check both cache and prefetch buffer
  - Only check head of prefetch buffer (buffer is a FIFO)
  - If buffer hit, move block into cache (avoids cache pollution)
- Can have multiple prefetch buffers: use LRU replacement
- Exploits spatial locality
  - Like having larger cache blocks without pollution problems
  - Only works for sequential access patterns
  - Works better for instructions than data, typically
  - Can increase memory traffic significantly (e.g., low accuracy)
Stride Predictor

Stride Prediction Table

<table>
<thead>
<tr>
<th>PC Tag</th>
<th>Prev. Address</th>
<th>Stride</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>LoadPC</td>
<td>R</td>
<td>Δ</td>
<td>1</td>
</tr>
</tbody>
</table>

Prefetch
C+Δ, C + 2Δ, …
Markov or Correlating Predictor

L1 Cache Miss Stream

Markov Table

<table>
<thead>
<tr>
<th>Miss Tag</th>
<th>Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addr. $a_1$</td>
<td>Addr. $a_2$</td>
</tr>
<tr>
<td>Addr. $a_2$</td>
<td>Addr. $a_3$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Addr. $a_N$</td>
<td>Addr. $a_1$</td>
</tr>
</tbody>
</table>

Miss: $a_1$

Prefetch: $a_2, a_3 \ldots, a_N$
Handling Aliasing: N\textsuperscript{th} Degree and N\textsuperscript{th} Order Predictors

- Higher Degree (Multiple Predictions)
- Higher Order (Multiple Tags)
Software Prefetching

- Allow compiler to control prefetching
  - Compiler determines what to prefetch
  - Compiler determines when to prefetch

- Less hardware support
  - No predictor
  - Typically prefetch into cache -> no prefetch buffer
  - Prefetch instruction: non-blocking load into R0

- Compiler inserts prefetch instructions into code
Software Prefetching for ArrayRefs

- Example: Dot Product

```
product = 0;
for (i=1; i <= N; i++)
    product += A[i] * B[i];
```

```
product = 0;
for (i=1; i <= N; i++) {
    prefetch(&A[i+PD]);
    prefetch(&B[i+PD]);
    product += A[i] * B[i];
}
```

- \(PD = \text{ceiling}(W / L)\)
  - \(PD\): prefetch distance, \(W\): work per iteration, \(L\): memory latency
Software Prefetching for Array Refs

- **Fix inefficiencies**
- Loop unrolling: isolate cache misses
- Software pipelining: handle first and last PD iterations separately

```c
product = 0;
for (i=1; i <= PD; i+=4) {  // prologue loop
    prefetch(&A[i]);
    prefetch(&B[i]);
}

for (i=1; i <= N-PD; i+=4) {  // steady-state loop
    prefetch(&A[i+PD]);
    prefetch(&B[i+PD]);
    product += A[i] * B[i];
    product += A[i+1] * B[i+1];
    product += A[i+2] * B[i+2];
    product += A[i+3] * B[i+3];
}

for ( ; i <= N; i++)          // epilogue loop
    product += A[i] * B[i];
```
Software Prefetching for Non-Striding References

- Indexed array references are prefetchable
- What about pointer chasing references?

```c
while(node) {
    prefetch(node->next);
    compute(node);
    node = node->next;
}
```

⇒ Pointer-Chasing Problem
Pointer Prefetching in Software

```c
while(node) {
prefetch(node->jump);
compute(node);
node = node->next;
}
```

Memory Parallelism!

But • Memory Overhead
• Runtime Overhead
• Significant Code Modification