

Parallelism

HY425 Lecture 14: Improving Cache Performance II

Dimitrios S. Nikolopoulos

University of Crete and FORTH-ICS

November 30, 2011

Overlapping memory latency with instructions

- ▶ Allow processor to work while memory serves requests
- ▶ Overlap memory latency with other instructions
- ▶ Essential in out-of-order processors

Hide memory latency with prefetching

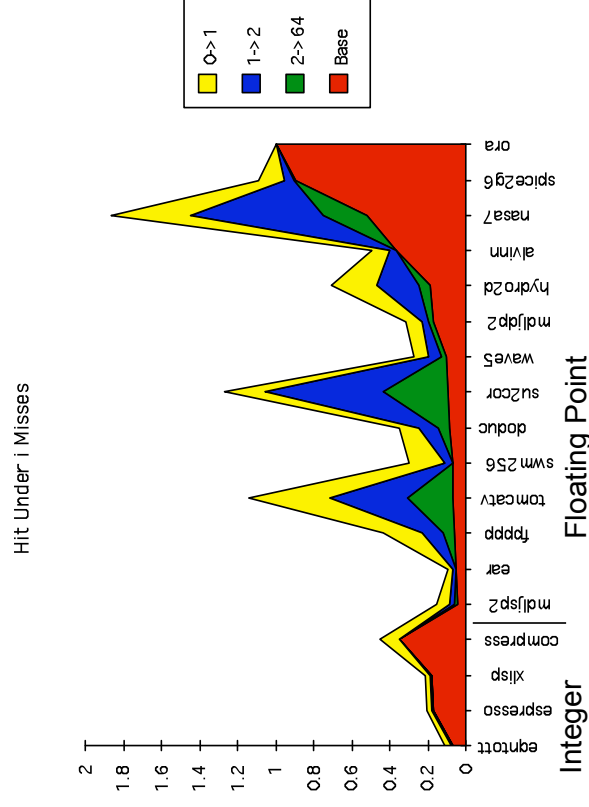
- ▶ Allow processor to prefetch extra data
- ▶ If data useful and fetched in time, reduces memory latency

Non-blocking caches

Hit under miss

- ▶ Processor continues execution while miss pending
- ▶ Cache serves hits while miss pending
- ▶ Need check for dependence violations
 - ▶ Logic for memory RAW, WAW, WAR, hazards
 - ▶ Dynamic issue processor
- ▶ Can overlap multiple misses
 - ▶ Requires suitable multi-bank, pipelined memory system
- ▶ Hard to measure miss penalty due to overlap of hits and misses

Hit under miss performance impact



Prefetching

Reduce memory latency

- ▶ Processor requests data in advance
 - ▶ Processor speculates that requested data will be accessed in the future
- ▶ Need a guess for what data will be needed in the future
- ▶ Guess easy in linear memory accesses
 - ▶ Fixed stride between data accesses
 - ▶ Strided prefetching
- ▶ Need prefetch triggers, e.g. two consecutive misses

Prefetching

Performance implications

- ▶ Prefetched data may **displace** other useful data from cache
- ▶ Prefetched data may come too early and be evicted before **used**
- ▶ Prefetched data may come too late and not be there when **needed**
- ▶ Prefetched data may be **useless**, i.e. not accessed at all
- ▶ Prefetched data **wastes** memory bandwidth, if useless

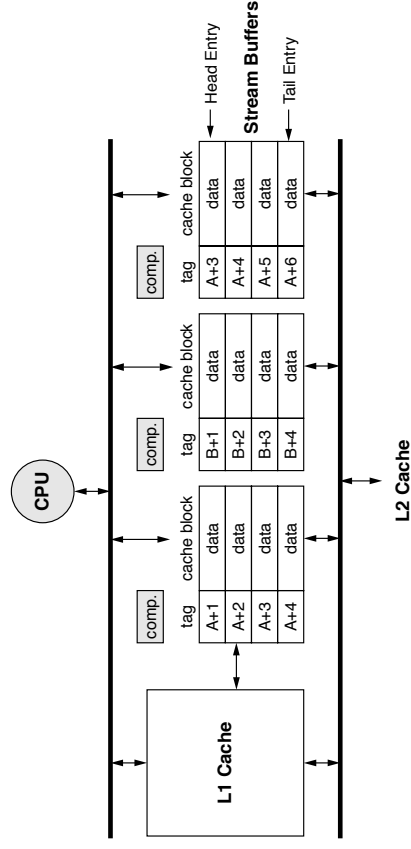
Online Prefetching Heuristics

- ▶ Prefetching helps codes with non-perfect spatial locality
- ▶ Can be initiated at any level of the memory hierarchy
- ▶ Prefetched data can be stored at any level of the memory hierarchy
- ▶ Branch prediction is a form of prefetching
 - ▶ Prefetching instructions ahead of their execution
- ▶ Simple lookahead prefetching
 - ▶ Prefetch block $i+1$ (or $i+2$, or $i+3, \dots$) upon demand fetching block i
 - ▶ Prefetch $i+1$ called **next-line prefetching**

Lookahead prefetching

- ▶ Three ways to prefetch:
 - ▶ Upon every memory access
 - ▶ Excessive memory traffic
 - ▶ Upon a cache miss
 - ▶ Catches contiguous streams of data non-present in the cache
 - ▶ Requires a miss before prefetching is initiated
- ▶ Tagged prefetch
 - ▶ One bit per cache block indicates if block was referenced
 - ▶ Bit set to zero if block was prefetched
 - ▶ Zero-to-one transition indicates access that **would have been** a miss if block were **not prefetched** and triggers lookahead prefetching

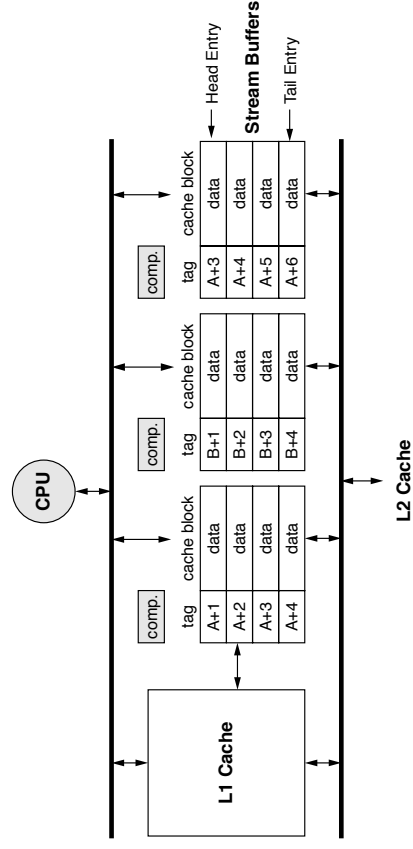
Stream Buffers



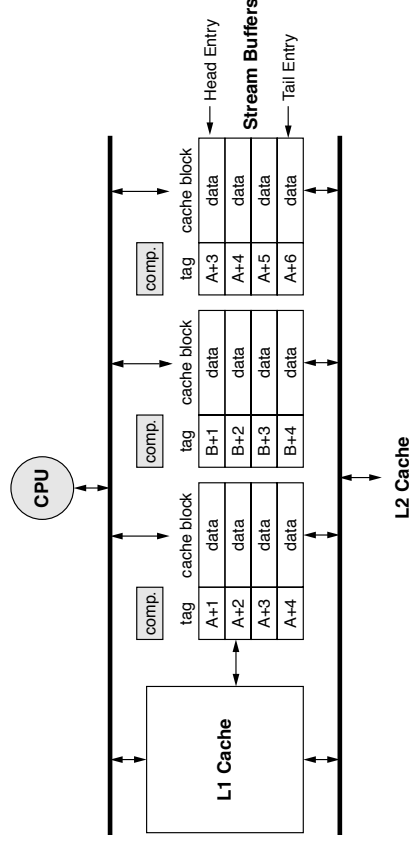
- ▶ Each buffer fetches data from one contiguous stream
- ▶ Cache and head entries of stream buffers checked upon access
- ▶ Cache miss may be served by head of stream buffer

Strided Prefetching Motivation

- ▶ Assume the reference pattern: A, B, A+2, B+1, A+4, B+2,...
- ▶ **Stream buffer space wasted in this case**

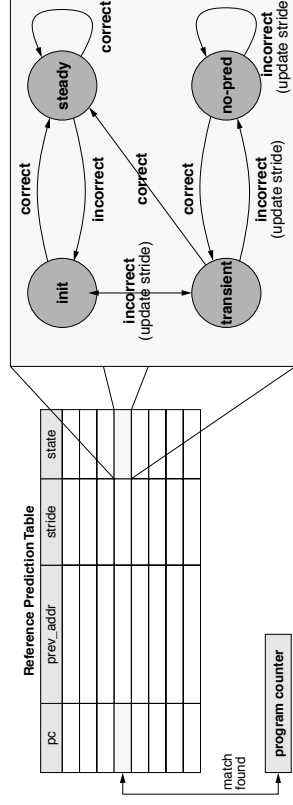


Stream Buffers



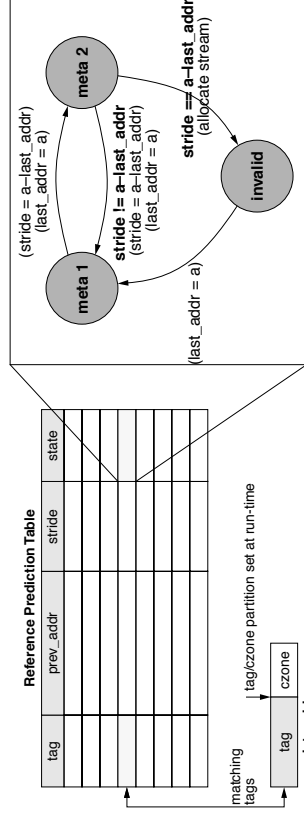
- ▶ If cache miss hits on stream buffer, head pointer moves down and prefetching is triggered
- ▶ Available bit per entry indicates if prefetching is in flight
- ▶ Buffer allocated when a stream of misses (e.g. address A, A+1, ...) is detected

Strided Prefetching



- ▶ **Reference prediction table** tracks load/store accesses and strides between addresses of the same load/store
- ▶ Stride used to make prediction of next address to prefetch
- ▶ FSM tracks and fixes stride predictions

Strided prefetching without the Program Counter

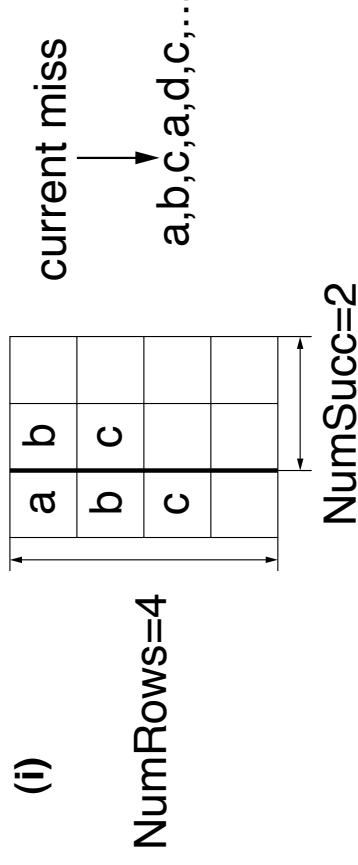


- ▶ Tag partitions the address space in regions
- ▶ Prefetcher separates streams in different address regions
- ▶ FSM requires three references that miss

Correlation Prefetching

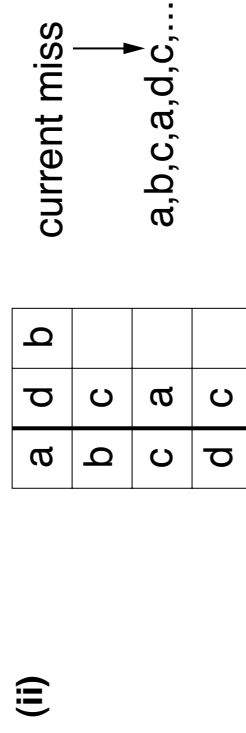
- ▶ Prefetching repeating patterns that are not dominated by a single or a few strides
- ▶ Algorithmic locality often not captured by strides (e.g. walking trees with dynamically allocated memory)

Correlation Table Miss Sequence



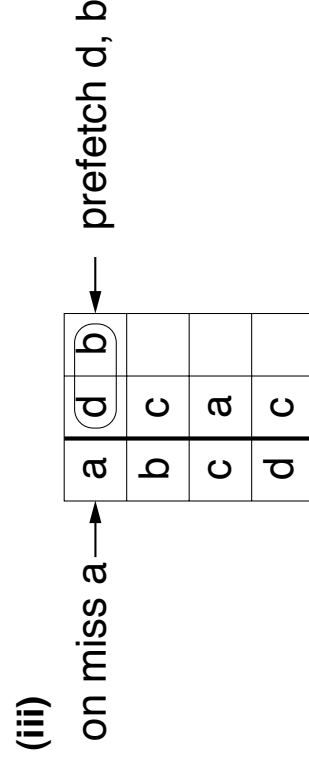
Correlation Prefetching

- ▶ Second Snapshot
- ▶ Successors of miss stored in MRU order
- ▶ Hardware keeps pointer to row of last observed miss
- ▶ On miss hardware updates successors and starts new correlation stream



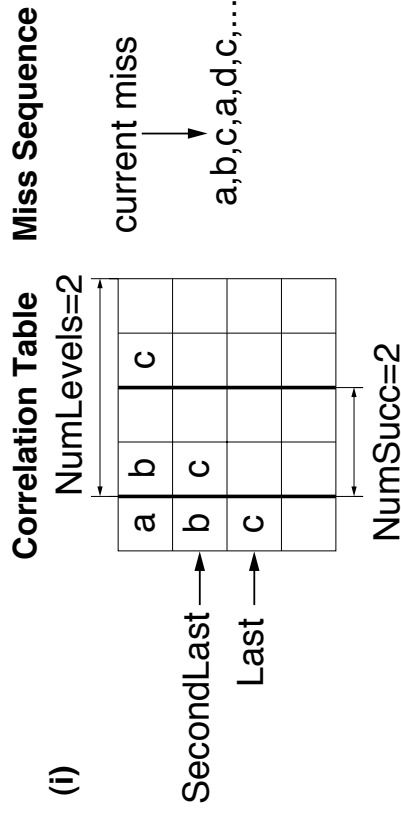
Correlation Prefetching

- ▶ Third Snapshot
- ▶ Only successors in single row prefetched, limits performance



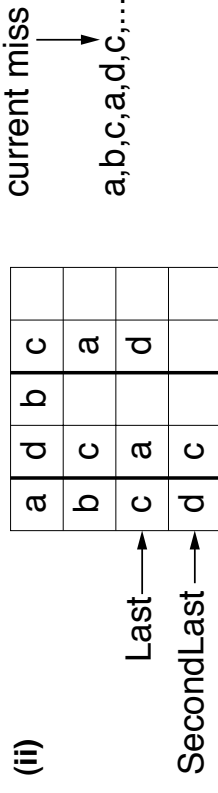
Multi-level Correlation Prefetching

- ▶ Maintain multiple pointers (a queue) in the correlation table
- ▶ Replicate miss information in the table



Multi-level Correlation Prefetching

- ▶ Maintain multiple pointers (a queue) in the correlation table
- ▶ Replicate miss information in the table

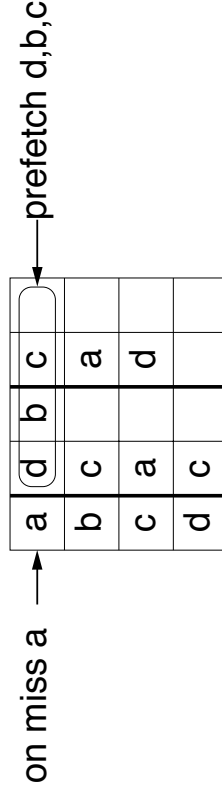


Multi-level Correlation Prefetching

- ▶ Use NLEVELS-1 successors to index the table and prefetch more
 - ▶ Example: NLEVELS=2
- (iii)
- | | | | |
|---|---|---|---|
| a | d | b | c |
| b | c | | a |
| c | a | | d |
| d | c | | |
- on miss a → prefetch d,b,c

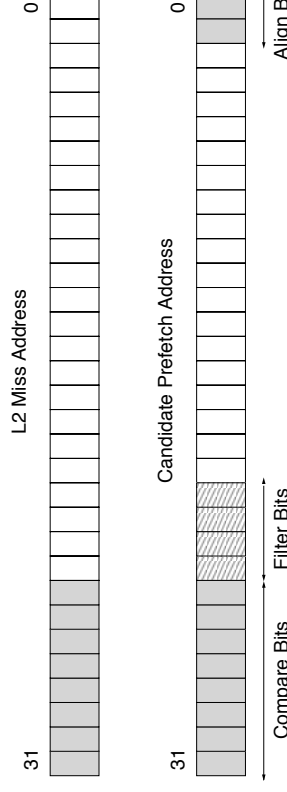
Multi-level Correlation Prefetching

- ▶ Content-based prefetching
 - ▶ Identify **two loads** out of which one produces the address consumed by the other
 - ▶ Indicates **pointer chasing**
- ▶ Content-directed prefetching
 - ▶ Scan words fetched from memory to **speculate if they are likely addresses**
 - ▶ Prefetch if word fetched is an address



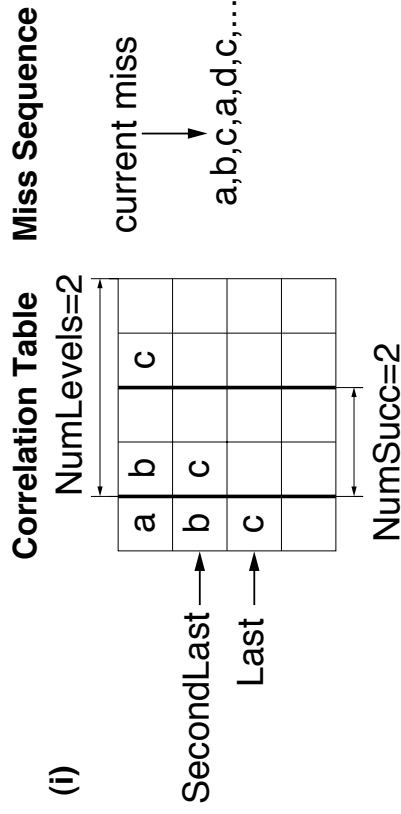
Other Prefetching Schemes

- ▶ Content-based prefetching
 - ▶ Identify **two loads** out of which one produces the address consumed by the other
 - ▶ Indicates **pointer chasing**
- ▶ Content-directed prefetching
 - ▶ Scan words fetched from memory to **speculate if they are likely addresses**
 - ▶ Prefetch if word fetched is an address



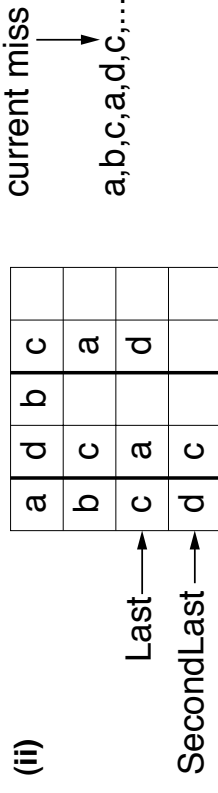
Multi-level Correlation Prefetching

- ▶ Maintain multiple pointers (a queue) in the correlation table
- ▶ Replicate miss information in the table



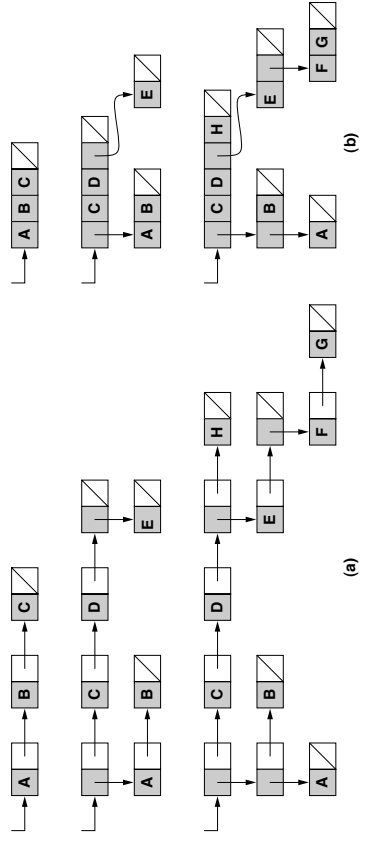
Multi-level Correlation Prefetching

- ▶ Maintain multiple pointers (a queue) in the correlation table
- ▶ Replicate miss information in the table

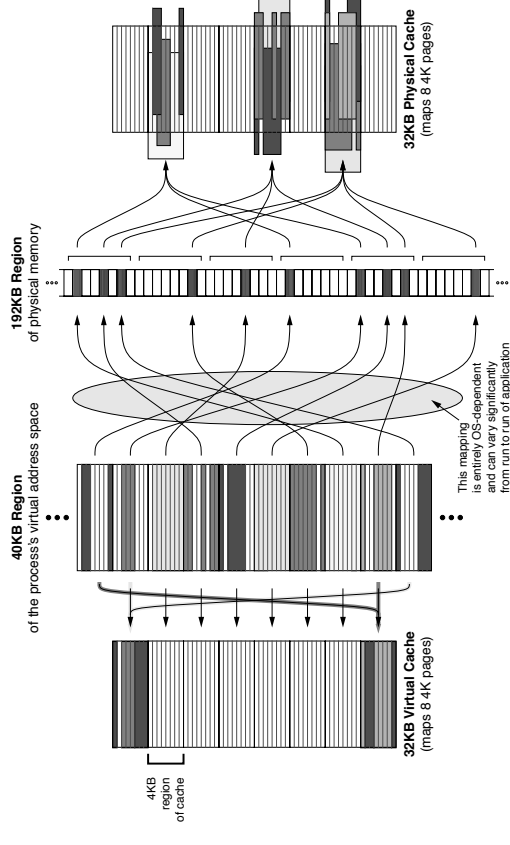


Dynamic Data Structure Reorganization for Locality

- ▶ List linearization
- ▶ Idea borrowed from garbage collection



Page Coloring



Page Coloring

- ▶ Physical layout of pages of data in DRAM differs from physical layout of pages of data in cache
- ▶ Consequence of **virtual-to-physical** address translation
- ▶ Software can implement **conflict-free placement of pages** if cache is **virtually-indexed**
 - ▶ Placing working set in adjacent virtual pages guarantees mapping in adjacent cache page-size regions
- ▶ Page coloring
 - ▶ Match **bottom bits of virtual page number with bottom bits of physical frame number**
 - ▶ Implemented using **bins of physical pages** that map to the same page-size region in the cache
 - ▶ Implementation lies in the operating system

Software Prefetching

Mowry's algorithm:

- ▶ Identify statically references that miss using **locality analysis**
- ▶ Examples assumes array elements are doubles and cache block is 32 bytes

```

a) for (j=2; j <= N-1; j++)
    for (i=2; i <= N-1; i++)
        A[i][j]=0.25*(B[i][i-1]+B[i][i+1]+B[j-1][i]+B[j+1][i]);

b) for (j=2; j <= N-1; j++)
    for (i=2; i <= N-PD; i+=4) { // Prologue
        prefetch(&B[i][0]);
        prefetch(&B[i-1][0]);
        prefetch(&B[i+1][0]);
        prefetch(&A[j][0]);
    }
    for (i=2; i < N-PD-1; i+=4) { // Steady State
        prefetch(&B[i][+PD]);
        prefetch(&B[i-1][+PD]);
        prefetch(&B[i+1][+PD]);
        prefetch(&A[j][+PD]);
    }
    A[i][j]=0.25*(B[i][i-1]+B[i][i+1]+B[j-1][i]+B[j+1][i]);
    A[i][i+1]=0.25*(B[i][i+2]+B[i-1][i+1]+B[i+1][i+1]);
    A[i][i+2]=0.25*(B[i][i+3]+B[i-1][i+2]+B[i+1][i+2]);
    A[i][i+3]=0.25*(B[i][i+4]+B[i-1][i+3]+B[i+1][i+3]);
    }
    for (i=N-PD; i <= N-1; i++) // Epilogue
        A[i][j]=0.25*(B[i][i-1]+B[i][i+1]+B[i-1][i]+B[i+1][i]);
    }
    
```


Software Prefetching

Mowry's algorithm:

- Algorithm identifies spatial reuse (all references in example), temporal reuse for each reference and temporal reuse for group of references
- Algorithm computes number of iterations between reuses and volume of data accessed to identify misses

```

a) for (j=2; j <= N-1; j++)
    for (i=2; i <= N-1; i++)
        A[i][j]=0.25*(B[i][i-1]+B[i][i+1]+B[j-1][i]+B[j+1][i]);

b) for (j=2; j <= N-1; j++) {
    for (i=2; i <= PD; i+=4) { // Prologue
        prefetch(&B[i][j]);
        prefetch(&B[i-1][j]);
        prefetch(&B[i+1][j]);
        prefetch(&A[i][j]);
    }
    for (i=2; i < N-PD-1; i+=4) { // Steady State
        prefetch(&B[i][j+PD]);
        prefetch(&B[i-1][j+PD]);
        prefetch(&B[i+1][j+PD]);
        prefetch(&A[i][j+PD]);
    }
    A[i][j]=0.25*(B[i][i-1]+B[i][i+1]+B[j-1][i]+B[j+1][i]);
    A[i][i+1]=0.25*(B[i][i]+B[i][i+2]+B[i-1][i+1]+B[j+1][i+1]);
    A[i][i+2]=0.25*(B[i][i+1]+B[i][i+3]+B[i-1][i+2]+B[j+1][i+2]);
    A[i][i+3]=0.25*(B[i][i+2]+B[i][i+4]+B[i-1][i+3]+B[j+1][i+3]);
}
for (i=N-PD; i <= N-1; i++) // Epilogue
    A[i][j]=0.25*(B[i][i-1]+B[i][i+1]+B[j-1][i]+B[j+1][i]);
}

```

Dimitrios S. Nikolopoulos

Reducing miss penalty or miss rate

HY425 Lecture 14: Improving Cache Performance II 26 / 29

Pointer Prefetching

Pointer chasing:

```

a) struct node {data, next}
    *ptr, *list_head;

    ptr = list_head;
    while (ptr) {
        prefetch(ptr->next);
        ...
        ptr = ptr->next;
    }

b) struct node {data, left, right}
    *ptr;

    void recurse(ptr) {
        prefetch(ptr->left);
        prefetch(ptr->right);
        ...
        recurse(ptr->left);
        recurse(ptr->right);
    }

```

Dimitrios S. Nikolopoulos

Reducing miss penalty or miss rate

HY425 Lecture 14: Improving Cache Performance II 26 / 29

Software Prefetching

Mowry's algorithm:

- Loop unrolling and peeling isolate cache misses (one per reference per unrolled iteration in example)
- Compiler inserts prefetches for references that miss using a prefetch distance, measured in iterations

```

a) for (j=2; j <= N-1; j++)
    for (i=2; i <= N-1; i++)
        A[i][j]=0.25*(B[i][i-1]+B[i][i+1]+B[j-1][i]+B[j+1][i]);

b) for (j=2; j <= N-1; j++) {
    for (i=2; i <= PD; i+=4) { // Prologue
        prefetch(&B[i][j]);
        prefetch(&B[i-1][j]);
        prefetch(&B[i+1][j]);
        prefetch(&A[i][j]);
    }
    for (i=2; i < N-PD-1; i+=4) { // Steady State
        prefetch(&B[i][j+PD]);
        prefetch(&B[i-1][j+PD]);
        prefetch(&B[i+1][j+PD]);
        prefetch(&A[i][j+PD]);
    }
    A[i][j]=0.25*(B[i][i-1]+B[i][i+1]+B[j-1][i]+B[j+1][i]);
    A[i][i+1]=0.25*(B[i][i]+B[i][i+2]+B[i-1][i+1]+B[j+1][i+1]);
    A[i][i+2]=0.25*(B[i][i+1]+B[i][i+3]+B[i-1][i+2]+B[j+1][i+2]);
    A[i][i+3]=0.25*(B[i][i+2]+B[i][i+4]+B[i-1][i+3]+B[j+1][i+3]);
}
for (i=N-PD; i <= N-1; i++) // Epilogue
    A[i][j]=0.25*(B[i][i-1]+B[i][i+1]+B[j-1][i]+B[j+1][i]);
}

```

Dimitrios S. Nikolopoulos

Reducing miss penalty or miss rate

HY425 Lecture 14: Improving Cache Performance II 27 / 29

Pointer Prefetching

Prefetching using jump pointers:

```

a) struct node {data, next, jump}
    *ptr, *list_head, *prefetch_array[PD], *history[PD];
    int i, head, tail;

    for (i = 0; i < PD; i++) // Prologue Loop
        prefetch(prefetch_array[i]);

    ptr = list_head;
    while (ptr->next) { // Steady State Loop
        prefetch(ptr->next);
        ...
        ptr = ptr->next;
    }

b) for (i = 0; i < PD; i++) history[i] = NULL;
    tail = 0;
    head = PD-1;

    ptr = list_head;
    while (ptr) { // Prefetch Pointer Generation Loop
        history[head] = ptr;
        if (history[tail])
            prefetch_array[tail] = ptr;
        else
            history[tail->jump] = ptr;
        head = (head+1)%PD;
        tail = (tail+1)%PD;
        ptr = ptr->next;
    }

```

Dimitrios S. Nikolopoulos

Reducing miss penalty or miss rate

HY425 Lecture 14: Improving Cache Performance II 27 / 29