

Cache Prefetching Techniques



SPARSH MITTAL

IIT HYDERABAD, INDIA

Motivation



- Memory system: crucial bottleneck in modern systems.
- We need latency hiding techniques
 - large caches: **area/energy penalty**
 - multithreading: **only useful for parallel apps**
 - prefetching: useful for both parallel and serial apps

Prefetching



- Used in nearly all high-performance processors
 - AMD Opteron, IBM Power8, Intel Xeon and Oracle Sparc M7 etc.
- Ideal case: prefetching can remove all cache misses [Ann01, Fer11]

Challenges of Prefetching



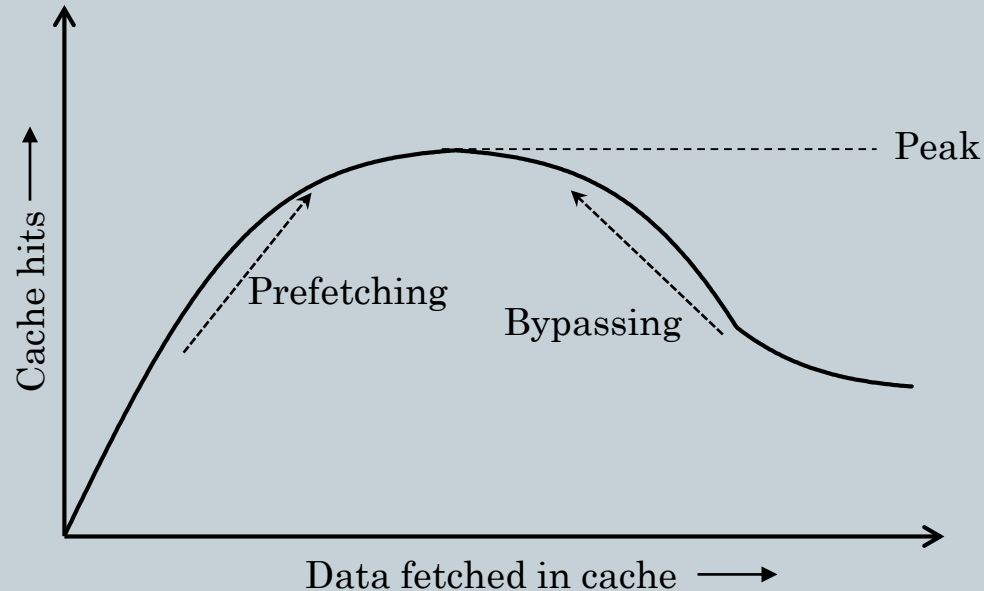
- Requires prediction of future access patterns: generally non-trivial
- Complex access patterns => sophisticated prefetchers required => huge metadata (tens of MBs) and latency overheads
- Storing metadata **off-chip** => requires frequent and costly communication of data on and off chip
- Storing metadata **on-chip** => only for small structures

Challenges of Prefetching



- Naïve prefetchers may bring useless lines which consume cache space and displace useful lines
- To avoid redundant prefetches, cache needs to be probed => energy waste
- Prefetching causes BW contention, esp. in multi-cores
- Reduction in misses brought by prefetching may not directly translate into performance improvement

Prefetching and bypassing are complementary



- Both prefetching and bypassing need to be balanced to improve performance

Classifications of prefetching



Hardware (HW) and Software (SW) prefetching



- **HW prefetching:**
 - Use additional storage to detect specific memory access patterns e.g., strided accesses.
 - Find addresses to prefetch based on this info
- **SW prefetching:** Insert prefetch instructions in source-code based on compiler/post-execution analysis

Hardware Prefetching

9

- Idea: Specialized hardware observes load/store access patterns and prefetches data based on past access behavior
- Tradeoffs:
 - + Can be tuned to system implementation
 - + Does not waste instruction execution bandwidth
 - More hardware complexity to detect patterns
 - Software can be more efficient in some cases

Software Prefetching

```
int x[N], y[N], z[N];  
for(int j=0; j<N; j++)  
{  
  
    z[j] = 3*x[j]+y[j];  
  
}
```

Original code

```
int x[N], y[N], z[N];  
for(int j=0; j<N/K; j++)  
{  
    PREFETCH(&x[j*K], K);  
    PREFETCH(&y[j*K], K);  
    for(int i=0; i<K; i++)  
    {  
        z[j*K+i] = 3*x[j*K+i]+y[j*K+i];  
    }  
}
```

Code with software prefetching instructions

Software prefetching

(+) Works well for very regular array-based access patterns

(-) Not easy for pointer-based data structures

Challenges of Software Prefetching



- Prefetch instructions take execution bandwidth
- Difficult to determine how early to prefetch
 - Prefetch distance depends on hardware implementation (memory latency, cache size, time between loop iterations)
→ portability reduced
 - Going too far back in code reduces accuracy (branches in between) but required since memory latency ~100s cycles

```
while (p) {  
    __prefetch(p→next);  
    work(p→data);  
    p = p→next;  
}
```

```
while (p) {  
    __prefetch(p→next→next→next);  
    work(p→data);  
    p = p→next;  
}
```

Which one is better?

X86 PREFETCH Instruction

PREFETCHh—Prefetch Data Into Caches


Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
OF 18 /1	PREFETCHT0 <i>m8</i>	Valid	Valid	Move data from <i>m8</i> closer to the processor using T0 hint.
OF 18 /2	PREFETCHT1 <i>m8</i>	Valid	Valid	Move data from <i>m8</i> closer to the processor using T1 hint.
OF 18 /3	PREFETCHT2 <i>m8</i>	Valid	Valid	Move data from <i>m8</i> closer to the processor using T2 hint.
OF 18 /0	PREFETCHNTA <i>m8</i>	Valid	Valid	Move data from <i>m8</i> closer to the processor using NTA hint.

Description


Fetches the line of data from memory that contains the byte specified with the source operand to a location in the cache hierarchy specified by a locality hint:

- T0 (temporal data)—prefetch data into all levels of the cache hierarchy.
 - Pentium III processor—1st- or 2nd-level cache.
 - Pentium 4 and Intel Xeon processors—2nd-level cache.
- T1 (temporal data with respect to first level cache)—prefetch data into level 2 cache and higher.
 - Pentium III processor—2nd-level cache.
 - Pentium 4 and Intel Xeon processors—2nd-level cache.
- T2 (temporal data with respect to second level cache)—prefetch data into level 2 cache and higher.
 - Pentium III processor—2nd-level cache.
 - Pentium 4 and Intel Xeon processors—2nd-level cache.
- NTA (non-temporal data with respect to all cache levels)—prefetch data into non-temporal cache structure and into a location close to the processor, minimizing cache pollution.
 - Pentium III processor—1st-level cache
 - Pentium 4 and Intel Xeon processors—2nd-level cache

microarchitecture
dependent
specification



different instructions
for different cache
levels



Advantages of SW Prefetching

HW prefetching

- Fails for very short streams
- When number of streams present in application exceed the HW resources, HW prefetchers may not clearly distinguish them
- Fails to detect loop bounds
- In commercial processors, HW prefetchers place data in lower caches (L2/L3) only

SW prefetching

- Good for short streams
- Can insert prefetch instructions for each stream individually
- Can easily ascertain loop bounds and avoid prefetching outside bounds
- Can place data directly in the right cache level

Advantages of HW Prefetching

HW prefetching

- Can account for runtime behavior and input variations
- Adapting aggressiveness easy
- Do not increase binary size
- No burden on programmer
- No need of adding prefetch instructions to ISA

SW prefetching

- Cannot account for these
- Difficult to control their aggressiveness
- Can greatly increase the instruction count (e.g., up to 100%)
- Burden on programmer
- Need to add extra instruct.

Using HW and SW Prefetching Together



- Perform prefetching for a larger variety of streams
- SW prefetch requests can be used to train the HW prefetcher.
- These prefetchers may interact negatively, e.g.,
 - SW prefetches inhibit the ability of HW to detect streams properly
 - Harmful prefetches brought by SW cause cache pollution and BW contention.

Data and instruction prefetching

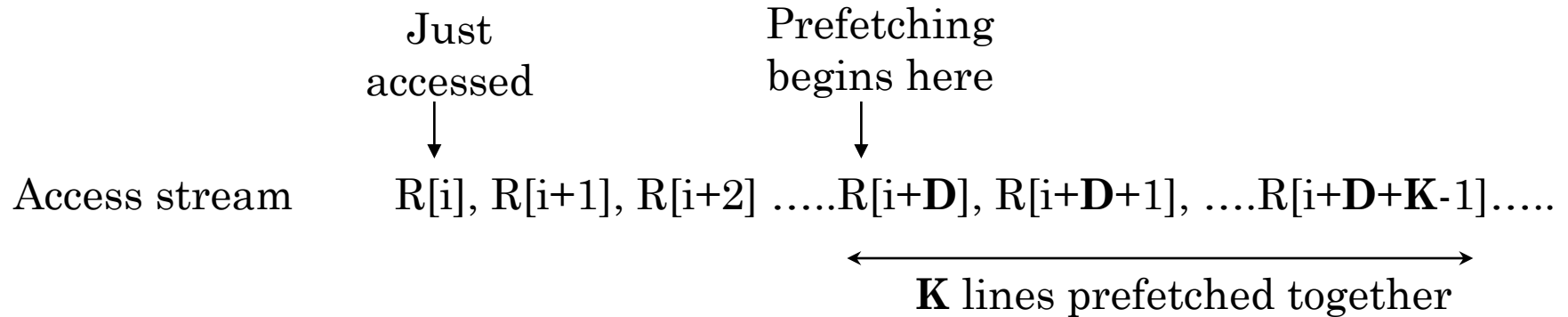


- **Data prefetching:** More challenging than instruction prefetching since data access pattern show less regularity and higher sensitivity to inputs
- **Instruction prefetching:** Not required for apps with negligible I-cache miss rate (e.g., scientific)
- Required for apps with large instruction working set size and high I-cache miss rate (e.g. commercial workloads)

Prefetching terminology/metrics and issues



Prefetch distance and degree



Prefetch distance = **D**
Prefetch degree = **K**

Useful terms



- **Coverage:** % of original misses eliminated by prefetched lines.
- **Accurate or useful prefetches:** those that eliminate original misses
- **Harmful prefetches:** those that induce misses by replacing useful data.
- **Redundant prefetch:** prefetched data already present in cache

Timeliness of Prefetch



- **Timely prefetch:** prefetched data placed in cache before they are accessed
- **Late prefetch:** prefetched data arrives after it is accessed => may not hide memory latency
- **Early prefetch:** prefetched data comes too early => it may be evicted before it is even used

Prefetching in Cache or Prefetch Buffers



Prefetching in Cache

- No need of checking a separate buffer
- May replace useful blocks => cache pollution
- Used in many systems: Intel Pentium 4, Core2, AMD systems, IBM POWER4,5,6

Prefetching in Buffer

- No pollution in cache
- Need to check both buffer and cache
- Need reorganization of chip architecture: what buffer size, where to place it, how to keep it coherent when to promote data to cache

Cache level where prefetching is done



- Cache level where prefetching is done:
 - Memory to L2, memory to L1
 - L2 to L1 (i.e., a separate prefetcher between levels)

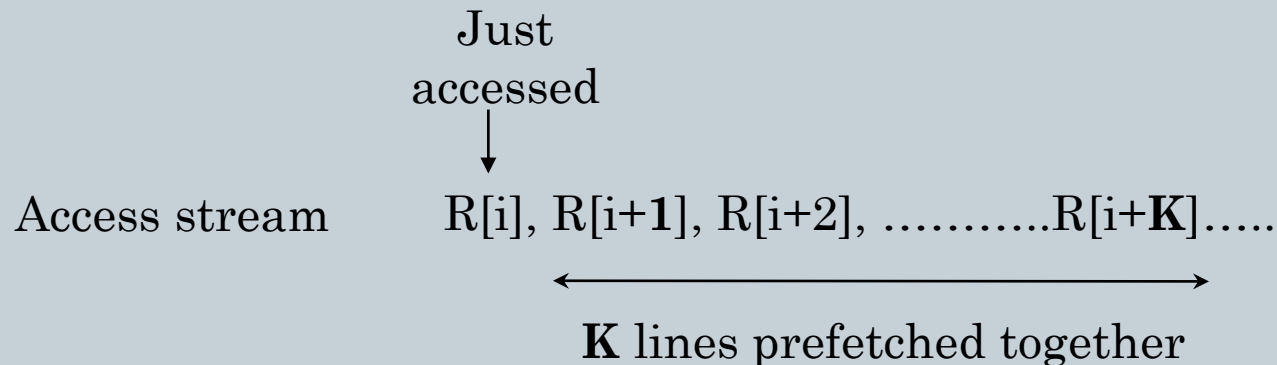
Examples of prefetching techniques



Next-K line Prefetcher



- “*Next-K*” line prefetcher brings next K lines after the current miss or access.



Stride Prefetcher

Prefetch lines showing strided pattern relative to the current miss

Access
stream

$R[i], R[i+Q], R[i+2Q], R[i+3Q]$

↑
Just accessed

Stride prefetcher

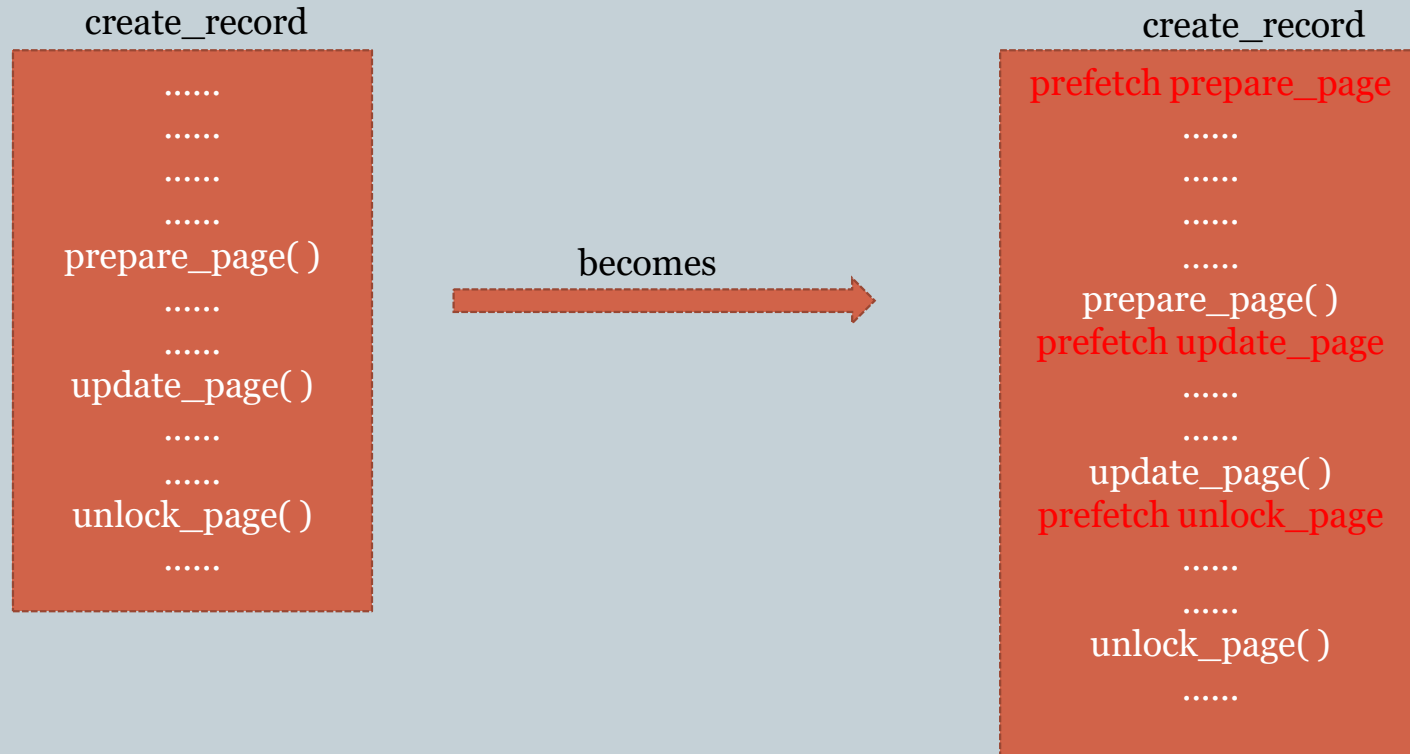
Stride = Q
Prefetch $R[(i+3Q+Q)]$

For $Q=1$, this is called stream prefetching.

Call Graph Prefetching



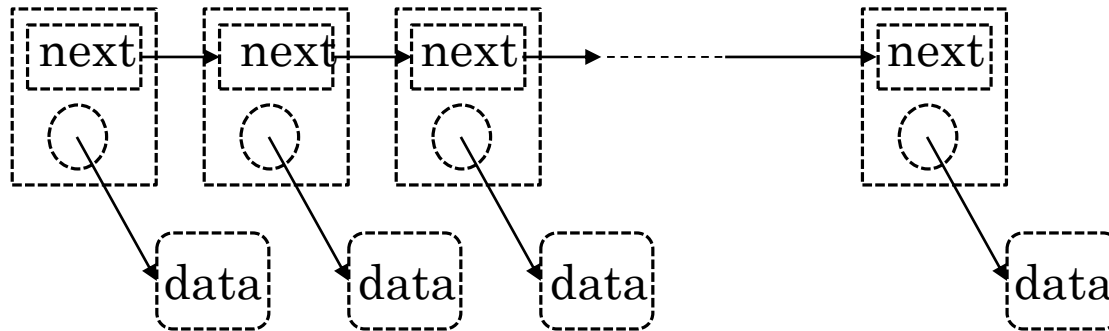
Key idea: Function access sequence has high repeatability



Examples of irregular access patterns

Some apps do not have regular patterns, e.g.

1. Linked data structure



2. Indirect indexing, e.g. $R[Z[i]]$

3. Hash function, e.g. $\text{hash}(i) \rightarrow R$

Complex prefetchers are required for these patterns

References



- S. Mittal, “A Survey of Recent Prefetching Techniques for Processor Caches”, ACM Computing Surveys, 2016 ([pdf](#))