

Prefetching (I)

Instructor: Josep Torrellas
CS533

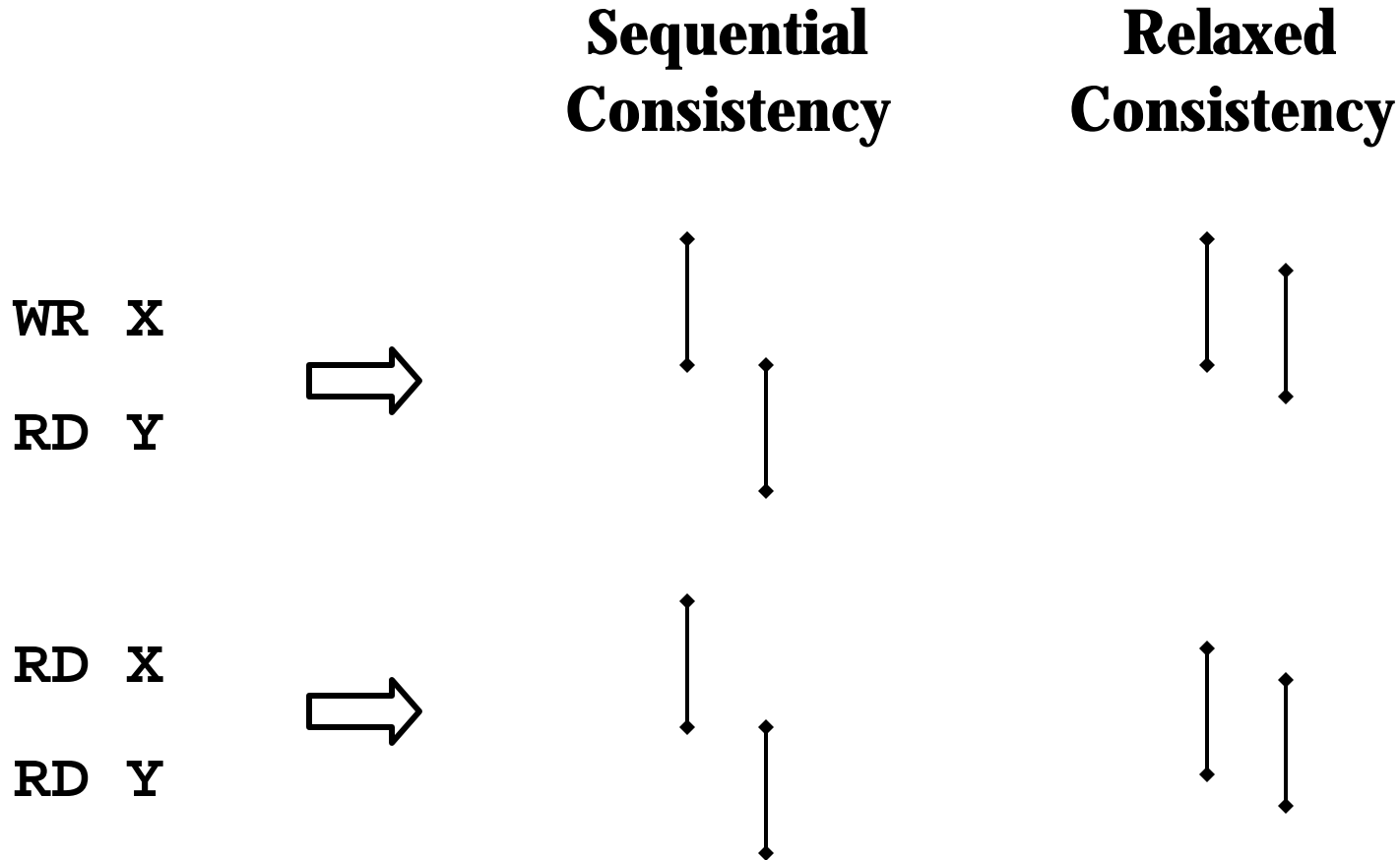


Latency Elimination/Hiding

- Caches and local memories
- Relaxed memory consistency models
- Prefetching/forwarding
- Multiple context processors



Relaxed Memory Consistency



Prefetching

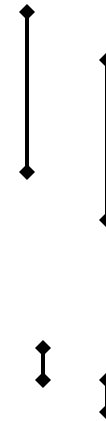
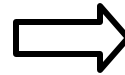
Prefetch X

Prefetch Y

...

RD X

RD Y




Prefetching Classification

- Binding vs. Non-binding
 - Binding: value of later “real” reference is bound when prefetch is performed
 - Restricts legal prefetch issue:
only when noone else can modify value
 - Additional high-speed storage needed (registers)
 - Non-binding: prefetch brings data closer, but value not bound until later “real” reference
 - Data remains visible to coherence protocol
 - Prefetch issue not restricted



Prefetching Classification

	Binding	Non-Binding
<code>prefetch(X)</code> <code>lock(L)</code> <code>X++</code> <code>unlock(L)</code>		

- Choice largely dictated by cache coherence
 - HW coherence is a requirement for non-binding



Prefetching Classification

- Software-controlled vs. Hardware-controlled



Prefetching initiated by processor executing a prefetch instruction

- **Programmer**
- **Compiler**



**HW prefetches at run-time.
No hints from SW.**

- **Instruction lookahead**
- **Long cache lines**



HW-Controlled Prefetching

- + Better dynamic information
- + No instruction overhead to issue prefetches
- Difficult to detect memory access patterns
- Lookahead limited by
 - branches
 - buffer size
- Long cache lines have false sharing
- Are “hard-wired” in the processor



SW-Controlled Prefetching

- + Extends possible prefetch-reference interval
- + Selectiveness based on program knowledge
- + Simplifies HW
- Requires sophisticated software intervention



Benefits of Prefetching

- Prefetch early enough
 - Completely hides latency
- Issue prefetches in blocks
 - Pipelining
 - Only first reference suffers
- Prefetch with ownership
 - Reduce write latency
 - Reduce network traffic



Compiler-Directed Prefetching: Issues

- What to prefetch – need to know what refs are going to hit or miss in the cache (since issuing prefetches has overheads)
 - Often not very difficult for programmer
 - Quite a challenging task for the compiler
 - When to prefetch – need to know how to schedule prefetches
 - Not too early (may get displaced from cache)
 - Not too late (data may not arrive in time)
- ⇒ Techniques such as software pipelining are critical



Definitions

- Coverage: Fraction of the original misses that are eliminated (partially or totally) by the prefetched lines
- Accuracy: Fraction of the prefetched lines that eliminate (partially or totally) original misses
- Can we get more misses after prefetching?
How?



Compiler-Directed Prefetching: SW

- Basic architecture support
 - Instructions for read and read-exclusive prefetches (plus exception model)
 - Lockup-free caches and associated machinery



Compiler Approach

- First use other tricks to get locality (e.g. blocking). Otherwise not fair.
 - Must understand data reuse already in code
 - Temporal, spatial and group reuse (See Mowry paper)
 - Need to understand it within and across loops
 - Determine what to prefetch based on the above
 - Optimize code so that address calculations, etc. have little overhead
 - Use loop splitting and loop unrolling
- ⇒ The above algorithm implemented in Stanford SUIF compiler



Problem: Unnecessary Prefetches

- Overhead of computing address and issuing prefetch
- Increased network traffic



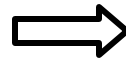
Reuse

- Temporal

```
DO j
  DO i
    ...
    =a[j]
```

- Spatial

```
DO i
  ...
  =a[i]
```



**Iterations i, i+1 use
same cache line**

- Group

```
DO i
  ...
  =a[i][0]+a[i+5][0]
```



Reducing Overhead: Example

```
prefetch(A[0]);
prefetch(A[1]);
prefetch(A[2]);
prefetch(A[3]);
for(i=0, i<n; i++){
    prefetch(A[i+4]);
    ..=a[i]
```

↓ **Use spatial locality**

```
prefetch(A[0]);
for(i=0, i<n; i++){
    if(i%4==0)
        prefetch(A[i+4]);
    ..=a[i]
```

**Suppose we need to
prefetch four ahead to
hide the latency**

Unroll



```
prefetch(A[0])
for(i=0, i<n; i+=4){
    prefetch(A[i+4])
    ..=a[i]
    ..=a[i+1]
    ..=a[i+2]
    ..=a[i+3]
```



Performance

- See Fig 3:
 - Prefetching applicable to many programs
 - Compiler eliminates much of the memory latency
 - Increase in the number of instructions
 - PF mem overhead:
 - When executing LD/ST and cache tags busy with a PF fill
 - When attempting to issue a PF and PF issue buffer full



Blocking + PF

- Blocking reduces the bandwidth demands of a program, therefore increasing the potential of PF
- However, fewer misses for PF to hide
- See Fig 8: blocking and prefetching together make programs more efficient (different case of GMTRY and VPENTA)

