Aside: Compiler Directed Coherence

- Compiler manages cache
  - No directory hardware (too complex, not scalable)
  - Divide program into computation units (epochs)

repeat /* computation of vector x_{i+1} = Ax_i + b */
odall j = 1 to N /* create N parallel tasks */
  xtemp[j] = b[j];
  for k = 1 to N
    xtemp[j] = xtemp[j] + A[j,k]*x[k];
  end doall

dodall j = 1 to N /* create N parallel tasks */
x[j] = xtemp[j]; /* write new vectors */
edoall /* end of epoch */
until all vectors computed
Compiler Directed Coherence (Continued)

- **Indiscriminate Invalidation**
  - Invalidate entire cache after each epoch
  - Uses uncached read/write as one option

- **Selective Invalidation**
  - Invalidate only cache lines that may introduce incoherence
  - Assumes write-through caches

- **Timestamp**
  - Clock per data structure incremented at end of epoch
  - Timestamp cache block with clock +1 when written
  - Valid if timestamp > clock

- **BUT POINTERS!**
- In general has only been shown to work for one word cache blocks

Latency Tolerance (Chapter 11)

- **Motivation**
  - Microprocessors getting faster faster than DRAM
  - DRAM access is now 100s of “instruction opportunities”
  - Worse for parallel machines

- **Of course**
  - Use memory hierarchy
  - Make each level as fast as possible
  - make each level better (e.g., bigger)

- **Eventually**
  - Must tolerate latency for additional parallelism
  - Old idea: multitasking for disk I/O
Latency Tolerance (Chapter 11)

- Motivation

Time in processor's critical path

\[ \alpha \alpha \alpha \alpha \alpha \alpha \]

\[ \beta \beta \beta \beta \beta \beta \]

(a) Baseline communication, no latency tolerance

Latency Tolerance for Message Passing

- Block Data Transfer
  - Of course -- messages

- Pre-communication
  - If possible
  - And must buffer

- Proceeding Past Long-Latency Events
  - Of course -- messages

- Multithreading within Node
  - Possible
  - But makes programming model more complex

(C) 2001 Mark D. Hill from Adve, Falsafi, Lebeck, Reinhardt, & Singh
Pre-Communication

\[
\begin{align*}
P_A & \quad P_B \\
& \quad \text{for } i \leftarrow 0 \text{ to } n-1 \text{ do} \\
& \quad \quad \text{compute } A[i]; \\
& \quad \quad \text{write } A[i]; \\
& \quad \quad \text{a_send }(A[i] \text{ to proc.P}_B); \\
& \quad \quad \text{end for} \\
& \quad \text{for } i \leftarrow 0 \text{ to } n-2 \text{ do} \\
& \quad \quad \text{a_receive }(myA[i+1] \text{ from } P_A); \\
& \quad \quad \text{while } (!\text{recv_probe}(myA[i])) \{ \\
& \quad \quad \quad \text{use } myA[i]; \\
& \quad \quad \} \\
& \quad \text{for } i \leftarrow 0 \text{ to } n-1 \text{ do} \\
& \quad \quad \text{compute } g(B[i]); \\
& \quad \quad \text{end for} \\
& \quad \text{while } (!\text{received}(myA[n-1])) \{ \\
& \quad \quad \text{use } myA[n-1]; \\
& \quad \quad \text{compute } g(B[n-1]) \\
\end{align*}
\]

Latency Tolerance Methods

(b) Single large message, no overlap with computation

(c) Single large message, overlap with computation
Latency Tolerance Methods

(d) Word-size overlapped messages, no overlap with computation

(e) Word-size overlapped messages, overlap with computation

(f) Word-size overlapped messages, overlap with computation, but only one outstanding message (no overlap with communication)

Latency Tolerance for Shared Memory 1 of 2

- Block Data Transfer
  - Harder to integrate
  - DMA engine (how virtualize?)
  - block load/store instructions
  - coalescing write buffer

- Pre-communication
  - Mostly prefetching
  - binding or non-binding (latter with coherence only)
  - initiated by hardware or software
    » what to prefetch
    » went to prefetch
  - TLB misses, page faults, & invalid exceptions
  - KSR-1 post-store -- a sender-initiated prefetch
  - Snarfing, deliver, update -- store allocation at destination hard
Latency Tolerance for Shared Memory 2 of 2

- Proceeding Past Long-Latency Events
  - Overlapping memory operations
  - Overlapping coherence operations
  - relaxed memory consistency models
    (or speculative implementation of strong model)

- Multithreading
  - Between instructions: HEP & Tera
  - On cache miss: MIT Alewife & IBM RS64 (RS/6000)
    - Few contexts for in-order core
  - Within cycle: Simultaneous Multithreading (SMT)
    - Good match to out-of-order core with register renaming
      (since backend need not know multiple threads exist)

Multithreading

(a) Single-threaded
(b) Multithreaded (interleaved)
(c) Simultaneous multithreaded