Caching

- Caches exploit the temporal and spatial locality of access exhibited by most programs
- Cache equation:

  $$Access\ Time_{Avg} = (1 - P) * Cost_{Hit} + P * Cost_{Miss}$$

  Where P consists of:

  – Compulsory misses

  – Capacity misses (size)

  – Conflict misses (associativity)

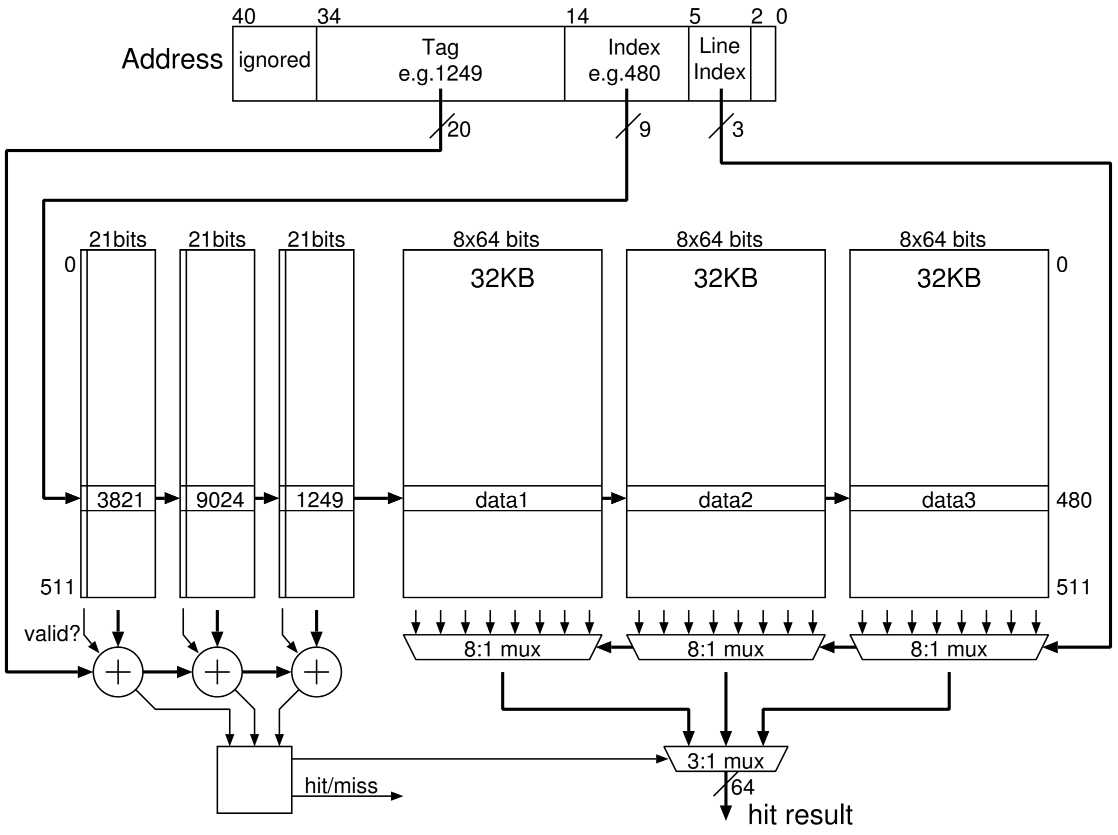
  – Coherence misses (multi-proc/DMA)

✘ Caches can increase $Cost_{Miss}$

- Build using fast (small and expensive) SRAM
- Tag RAM and Data RAM

Associativity

- Direct Mapped (1-way, no choice)
  - potentially fastest: tag check can be done in parallel with speculatively using data
- *n*-way Set Associative (choice of *n* e.g. 2/4/8)
- Fully associative (full choice)
  - many-way comparison is slow



A 96KB 3-way set associative cache with 64 byte lines (supporting 2^{35} bytes of cacheable memory)

Replacement Policy

- Associative caches need a replacement policy
- FIFO
  - ✘ Worse than random
- Least Recently Used (LRU)
  - ✘ Expensive to implement
  - ✘ Bad degenerate behaviour
    - ∗ sequential access to large arrays
- Random

- Use an LFSR counter

  ✔ No history bits required

  ✔ Almost as good as LRU

  ✔ Degenerate behaviour unlikely

- Not Last Used (NLU)
  - Select randomly, but NLU

  ✔ $log_2 n$ bits per set

  ✔ Better than random

Caching Writes

- Write-Back *vs.* Write Through
- Read Allocate *vs.* Read/Write Allocate
- Allocate only on reads and Write-Through
  - Writes update cache only if line already present
  - All writes are passed on to next level
  - Normally combined with a *Write Buffer*
- Read/Write Allocate and Write-Back
  - On write misses: allocate and fetch line, then modify
  - Cache holds the only up-to-date copy
  - Dirty bit to mark line as modified
  - ✔ Helps to reduce write bandwidth to next level
  - Line chosen for eviction may be dirty
    - *Victim writes* to next level
    - need fast victim writes to avoid double latency read
    - e.g. write victim, read new line, modify

        * exclusive: swap with next level

- May have different policies depending on special instructions or TLB entries

Write Buffers

- Small high-bandwidth resource for receiving store data

- Give reads priority over writes to reduce load latency
  - All loads that miss must check write buffer
  - If RaW hazard detected:
    - ∗ flush buffer to next level cache and replay
    - ∗ or, service load from buffer (PPro, 21264)

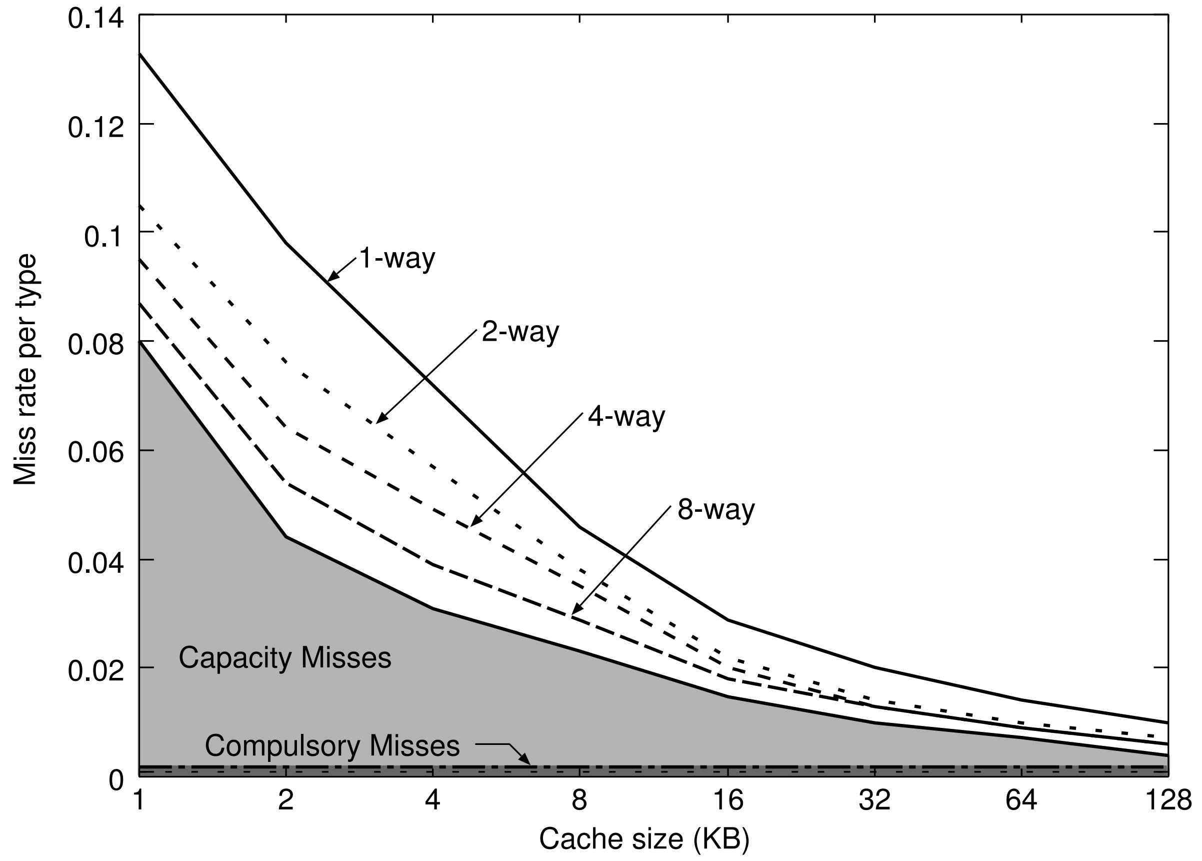
- *Merge* sequential writes into a single transaction

- *Collapse* writes to same location

- Drain write buffer when bus otherwise idle
- 21164: 6 addresses, 32 bytes per address
- ARM710: 4 addresses, 32 bytes total

Cache Miss Rate Example.

- SPEC 92 on MIPS
- 32 byte lines
- LRU replacement
- Write allocate/write back



A direct-mapped cache of size N has about the same miss rate as a 2-way set-associative cache of size $N/2$

L1 Caches

- L1 I-cache
  - Single-ported, read-only (but may snoop)
  - Wide access (e.g. block of 4 instrs)
  - (trace caches)

- L1 D-cache
  - Generally 8-64KB 1/2/4-way on-chip
    - Exception: HP PA-8200
  - Fully pipelined, Low latency (1-3cy), multi-ported
  - Size typ constrained by propagation delays

- Trade miss rate for lower hit access time
  - May be direct-mapped
  - May be write-through
- Often virtually indexed
  - Access cache in parallel with TLB lookup
  - Need to avoid virtual address aliasing
    - Enforce in OS
    - or, Ensure index size $<$ page size
      (add associativity)

Enhancing Performance :1

- Block size (Line size)
  - Most currently 32 or 64
  - ✔ Increasing block size reduces # compulsory misses
  - ✔ Typically increases bandwidth
  - ✘ Can increase load latency and # conflict misses

- Fetch critical-word-first and early-restart
  - Return requested word first, then wrap
  - Restart execution as soon as word ready
  - ✔ Reduces missed-load latency

  - Widely used. Intel order *vs.* linear wrap

- Nonblocking caches
  - Allow hit under miss (nonblocking loads)
  - Don't stall on first miss: allow multiple outstanding misses
    - merge misses to same line
  - Allow memory system to satisfy misses out-of-order

  ✔ Reduces effective miss penalty

Enhancing Performance :2

- Victim caches
  - Small highly associative cache to backup up a larger cache with limited associativity
  - ✔ Reduces the cost of conflict misses

- Victim buffers
  - A small number of cache line sized buffers used for temporarily holding dirty victims before they are written to L_{n+1}
  - Allows victim to be written *after* the requested line has been fetched
  - ✔ Reduces average latency of misses that evict dirty lines

- Sub-block presence bits
  - Allows size of tag ram to be reduced without increasing block size
  - Sub-block dirty bits can avoid cache line fills on write misses
    - (would break coherence on multiprocessors)

L2 caches

- L2 caches help hide poor DRAM latency
  - large write-back cache

- L2 caches used to share the system bus pins (e.g. Pentium)
  - ✘ electrical loading limits performance

- now, a dedicated ‘backside bus’ is used

- L2 on same die (21164)
  - ✔ low latency and wider bus
  - ✔ associativity easier

  - ✘ limited die size, so may need an L3
    - ∗ (e.g. 21164 has 2-16MB L3)

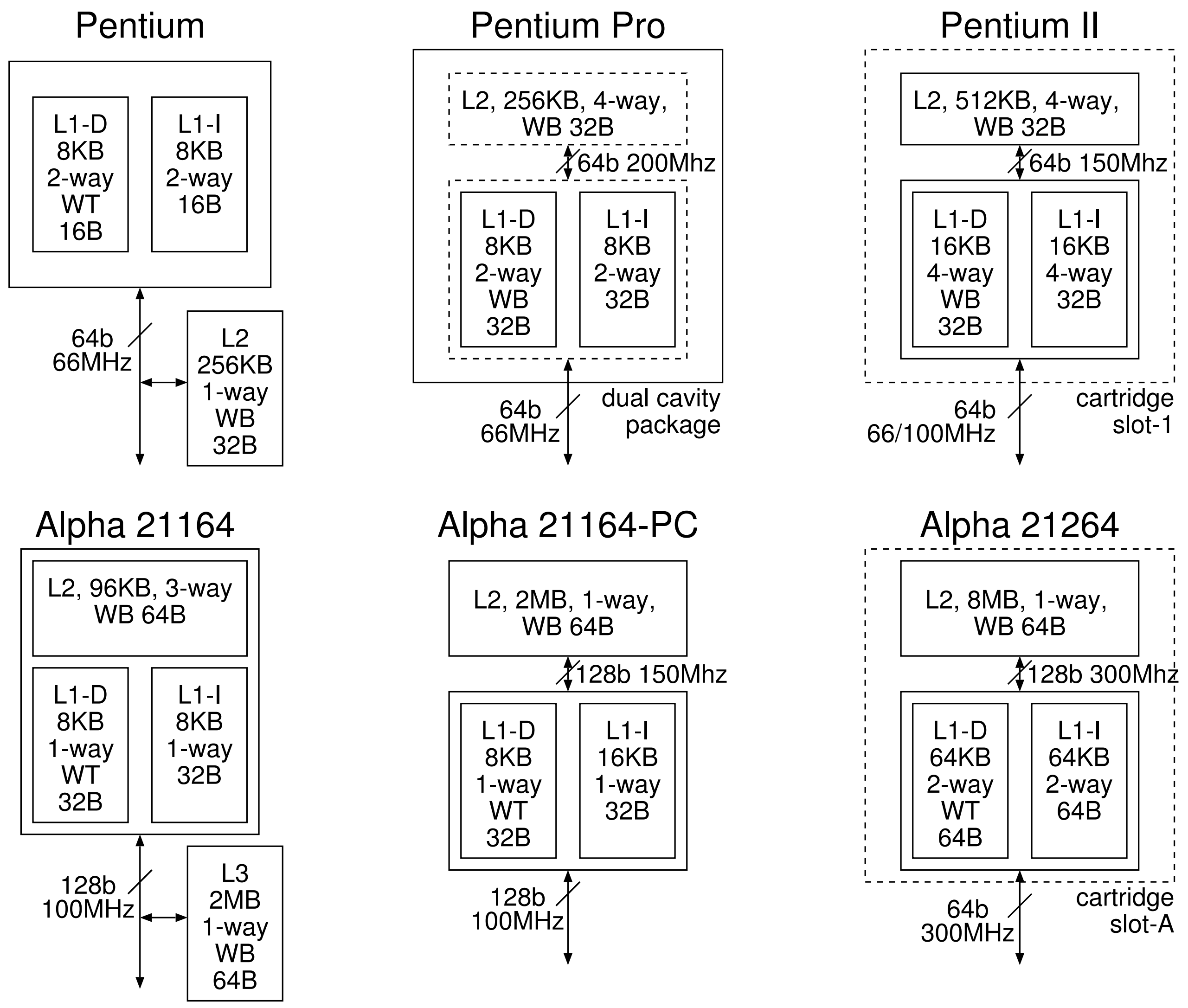
- L2 in CPU package (Pentium Pro)
  - ✔ lower latency than external

- L2 in CPU 'cartridge' (Pentium II)
  - ✔ controlled layout
  - ✔ use standard SSRAM

- L2 on motherboard
  - ✘ requires careful motherboard design

- L1/L2 inclusive *vs.* exclusive

Hierarchy Examples



Performance Examples

Ln	size	n-way	line	write	lat(cy)	lat(ns)	ld(MB/s)	st(MB/s)
L1	8KB	1	32	WT	1	4	1205	945
L2	96KB	3	64	WB	7	26	654	945
L3	2048KB	1	64	WB	22	83	340	315
MM	-	-	-	-	96	361	140	113

266MHz

21164 EB164 (Alcor/CIA)

Ln	size	n-way	line	write	lat(cy)	lat(ns)	ld(MB/s)	st(MB/s)
L1	8KB	2	16	WT	1	5	634	82
L2	256KB	1	32	WT	11	55	193	82
MM	-	-	-	-	28	140	123	81

200MHz

Pentium 430HX

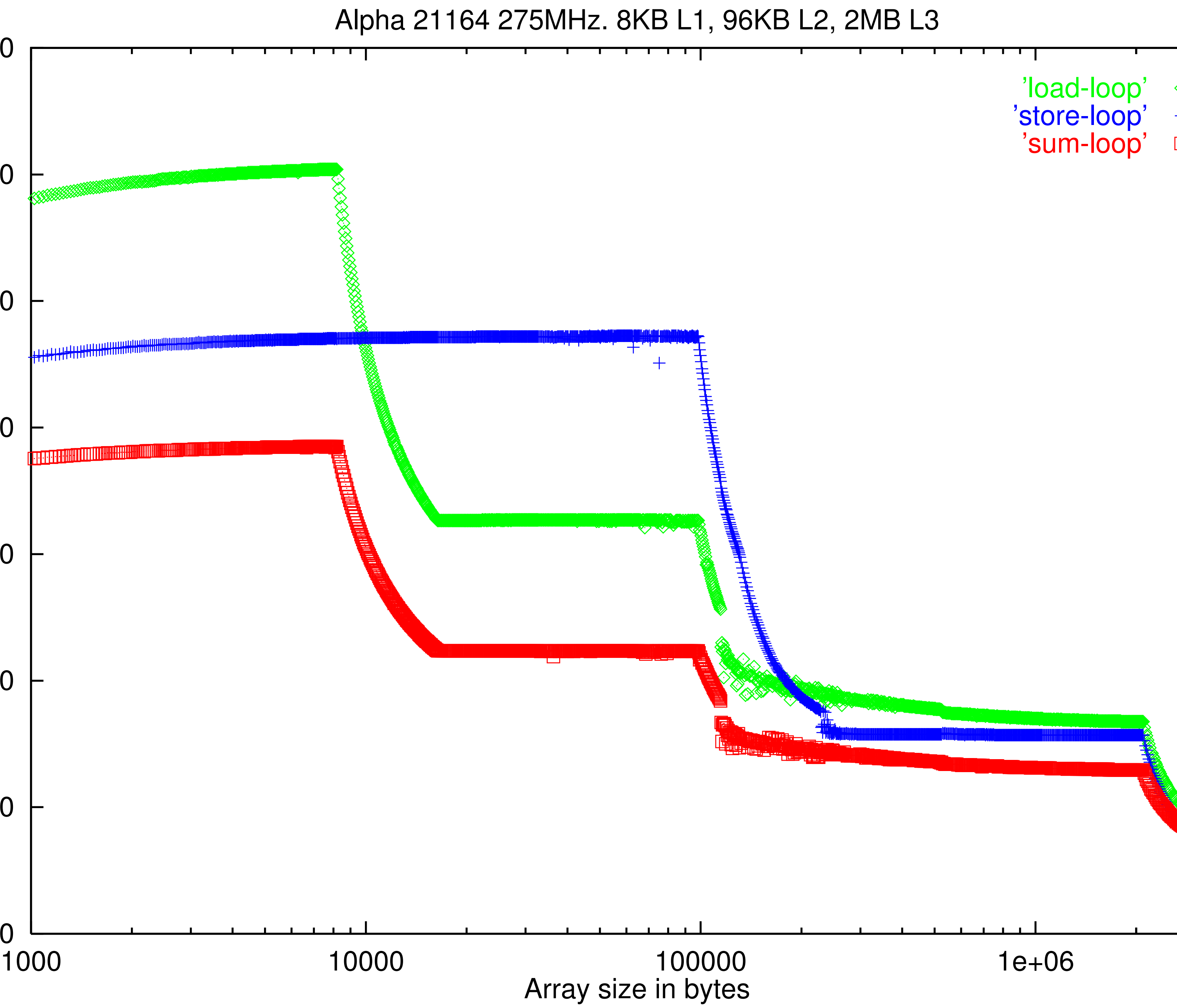
Ln	size	n-way	line	write	lat(cy)	lat(ns)	ld(MB/s)	st(MB/s)
L1	8KB	2	32	WB	2	10	695	471
L2	256KB	4	32	WB	6	30	426	426
MM	-	-	-	-	44	220	179	87

200MHz PPro 440FX

Ln	size	n-way	line	write	lat(cy)	lat(ns)	ld(MB/s)	st(MB/s)
L1	16KB	4	32	WB	2	7	1048	735
L2	512KB	4	32	WB	15	50	545	282
MM	-	-	-	-	64	213	231	116

300MHz PII

440LX



Alpha 21164 275MHz. 8KB L1, 96KB L2, 2MB L3
'load-loop'
'store-loop'
'sum-loop'
1000
10000
100000
1e+06
Array size in bytes

L3 Caches

- A third level of cache on chip.
- Use a denser memory technology than L2.
- Just a way to fill up silicon ?
- Perhaps moves bottleneck to TLB misses.

Main Memory

- Increasing sequential bandwidth
  - Wide memory bus
  - Interleave memory chips

  $\Rightarrow$ DDR SDRAM or RAMBUS
- Access latency can impair bandwidth
  - Larger cache block sizes help
- Reducing average latency
  - Keep memory banks ‘open’
    - $\ast$ Quick response if next access is to same DRAM Row

- Multiple independent memory banks
  - Access to an open row more likely
  - SDRAM/RAMBUS chips contain multiple banks internally
- System bus that supports multiple outstanding transaction requests
  - Service transactions out-of-order as banks become ready

Programming for caches

- Design algorithms so working set fits in cache
  - Large lookup tables may be slower than performing the calculation
- Organise data for spatial locality
  - Merge arrays accessed with the same index
- Fuse together loops that access the same data
- Prefer sequential accesses to non-unit strides
  - innermost loop should access array sequentially
- If row and column access to 2D arrays is necessary, use *cache blocking*
  - divide problem into sub-matrices that fit cache
  - e.g. matrix multiply $C = C + A \times B$

```
for (kb=0;kb<N;kb+=b){
  for (jb=0;jb<N;jb+=b){
    for (ib=0;ib<N;ib+=b){
      for(k=kb;k<kb+b;++k){
        for(j=jb;j<jb+b;++j){
          for(i=ib;i<ib+b;++i){
            C[k][i] = C[k][i] + ( A[k][j] * B[j][i] );
    } } } } } }
```

- Avoid access patterns that are likely to cause conflict misses (aliasing)
  - e.g. large powers of 2
- Large strides can thrash the TLB

Special Instructions

- Prefetch
  - fetch data into L1, suppressing any exceptions
  - enables compiler to speculate more easily
    e.g. Alpha: `ld r0 ← [r1]`
- ‘Two-part loads’ (e.g. IA-64)
  - speculative load suppresses exceptions
  - ‘check’ instruction collects any exception
  - enables compiler to ‘hoist’ loads to as early as possible, across multiple basic blocks

  -
    ```
    ld.s r4 ← [r5]
    chk.s r4
    ```
- Load with bypass hint
  - indicates that the load should bypass the cache, and thus not displace data already there
  - e.g. random accesses to large arrays
- Load with spatial-locality-only hint (non-temporal)

  - fetch line containing the specified word into a special buffer aside from the main cache
    * or, into set's line that will be evicted next

- Store with spatial-locality-only hint (non-temporal)

- Write invalidate
  - allocate a line in cache, & mark it as modified
  - avoids mem read if whole line is to be updated