Lecture 7:
Memory Hierarchy—3 Cs and 7 Ways to Reduce Misses

Professor David A. Patterson

Computer Science 252

Fall 1996

Vector Summary

- Like superscalar and VLIW, exploits ILP
- Alternate model accomodates long memory latency
- Requires memory performance as well as many pipelined functional units and registers
- Much easier for hardware: more powerful instructions, more predictable memory accesses, fewer branches, ...
- Multimedia instructions (Intel MMX, SPARC Viz) represent a resurgence of vector-like instructions
- What % of computation is vectorizable? What % do compilers deliver? For new apps?

Vector: CVI instruction

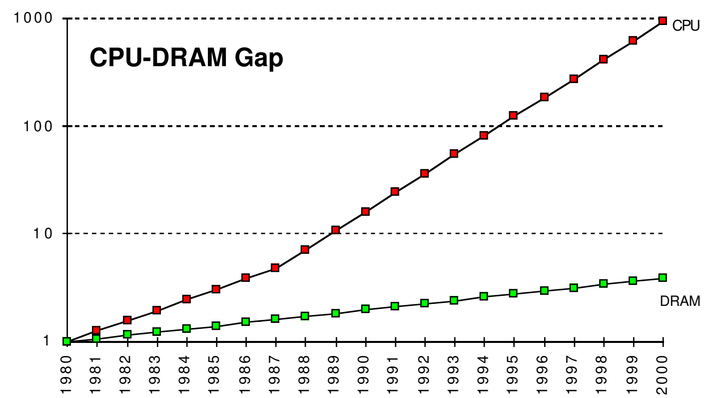
- Use `CVI` to create index 0,m, 2m, ..., 63m
- One use: create 0, 1, 2 ... to vectorize "`A[i]+i`"
- Compresses out elements where zeros in mask bits
  - Mask = "110101…" => `CVI V2,#8` generates 0,8,24,40,…
- Used for conditional code to compress vector (B-28):

```
SNESV   F0,V1 ;  sets vector mask bits to 1 if V1i ≠ F0
CVI     V2,#8 ;  creates indices in V2
POP     F0,VM ;  R1 = number of 1s in vector mask
MOVI2S  VLR,R1;  sets mask bits to 1 if V1i ≠ F0
CVM           ;  clear vector mask
LVI V3,(Ra+V2);  Load A elements ≠ 0 in original mask
LVI V4,(Rb+V2);  Load B elements ≠ 0 in original mask
SUBV V3,V3,V4 ;  A(I)=A(I)-B(I) when ≠ 0 in old mask
SVI V3,(Ra+V2);  Store A elements ≠ 0 in original mask
```

- For conditional expressions, what branch frequency faster for gather/scatter vs. masked vector?

Review: Who Cares About the Memory Hierarchy?

- Processor Only Thus Far in Course:
  - CPU cost/performance, ISA, Pipelined Execution



- 1980: no cache in µproc; 1995 2-level cache, 60% trans. on Alpha 21164 µproc (150 clock cycles for a miss!)

Review: Four Questions for Memory Hierarchy Designers

- Q1: Where can a block be placed in the upper level? *(Block placement)*
  - Fully Associative, Set Associative, Direct Mapped
- Q2: How is a block found if it is in the upper level? *(Block identification)*
  - Tag/Block
- Q3: Which block should be replaced on a miss? *(Block replacement)*
  - Random, LRU
- Q4: What happens on a write? *(Write strategy)*
  - Write Back or Write Through (with Write Buffer)

Review: Cache Performance

CPU time = (CPU execution clock cycles + Memory stall clock cycles) x clock cycle time

Memory stall clock cycles = (Reads x Read miss rate x Read miss penalty + Writes x Write miss rate x Write miss penalty)

Memory stall clock cycles = Memory accesses x Miss rate x Miss penalty

Review: Cache Performance

CPUtime = IC x ($CPI_{execution}$ + Mem accesses per instruction x Miss rate x Miss penalty) x Clock cycle time

Misses per instruction = Memory accesses per instruction x Miss rate

CPUtime = IC x ($CPI_{execution}$ + Misses per instruction x Miss penalty) x Clock cycle time

Review: Improving Cache Performance

1. Reduce the miss rate,

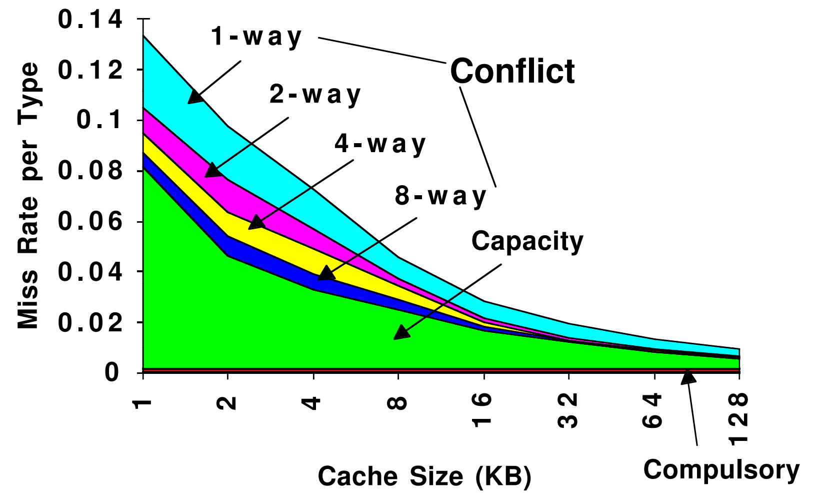
2. Reduce the miss penalty, or

3. Reduce the time to hit in the cache.

Reducing Misses

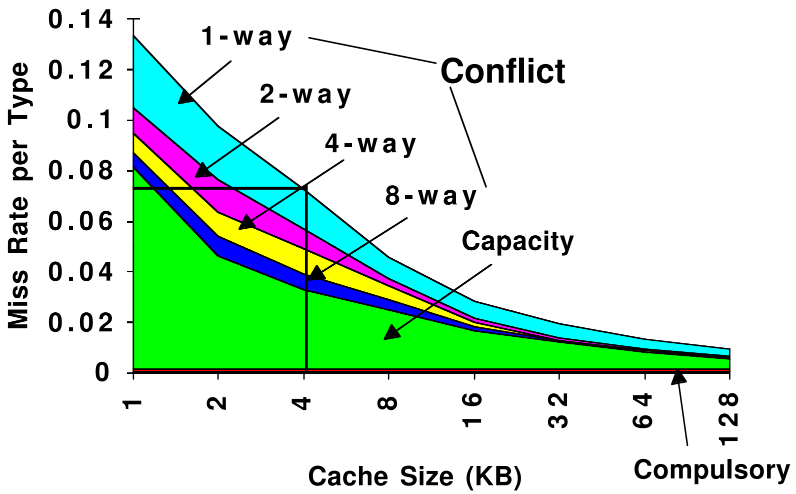
- **Classifying Misses: 3 Cs**
  - ***Compulsory*—The first access to a block is not in the cache, so the block must be brought into the cache. These are also called *cold start misses* or *first reference misses*. *(Misses in Infinite Cache)***
  - ***Capacity*—If the cache cannot contain all the blocks needed during execution of a program, capacity misses will occur due to blocks being discarded and later retrieved. *(Misses in Fully Associative Size X Cache)***
  - ***Conflict*—If the block-placement strategy is set associative or direct mapped, conflict misses (in addition to compulsory and capacity misses) will occur because a block can be discarded and later retrieved if too many blocks map to its set. These are also called *collision misses* or *interference misses*. *(Misses in N-way Associative, Size X Cache)***

3Cs Absolute Miss Rate

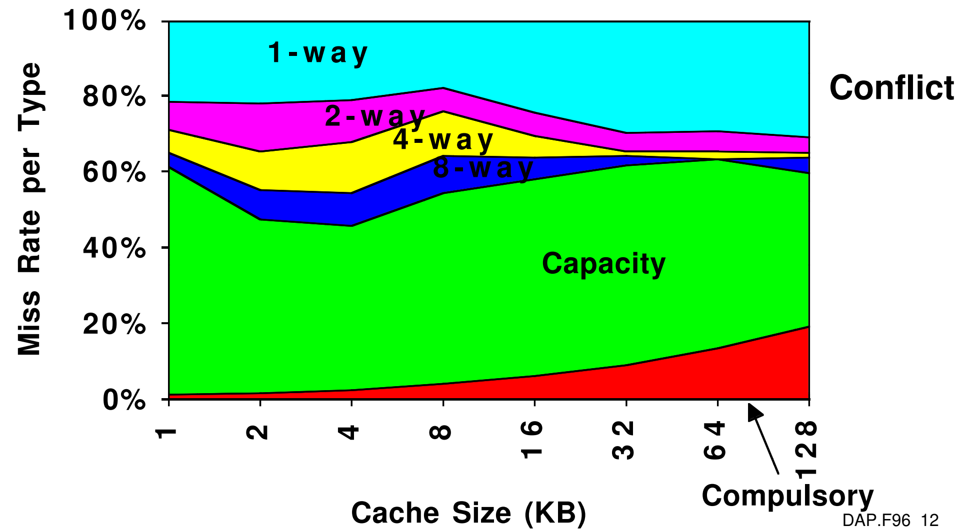


2:1 Cache Rule

miss rate 1-way associative cache size X
= miss rate 2-way associative cache size X/2



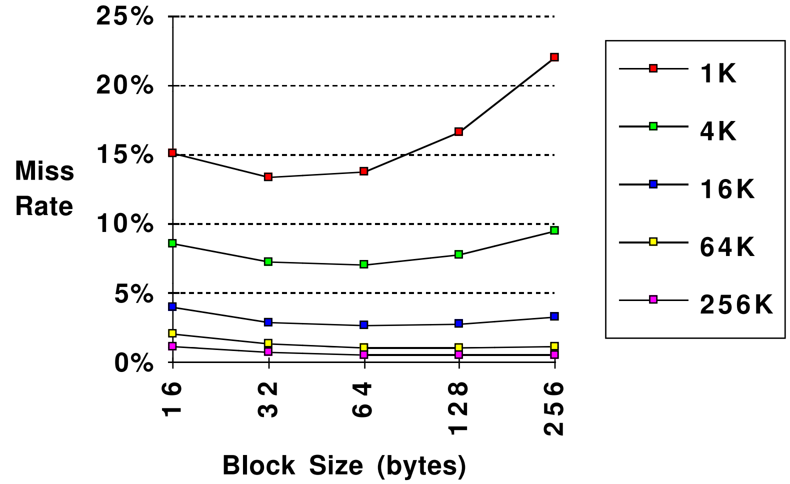
3Cs Relative Miss Rate



How Can Reduce Misses?

- **Change Block Size? Which of 3Cs affected?**
- **Change Associativity? Which of 3Cs affected?**
- **Change Compiler? Which of 3Cs affected?**

1. Reduce Misses via Larger Block Size



2. Reduce Misses via Higher Associativity

- **2:1 Cache Rule:**
  - **Miss Rate DM cache size N ≈ Miss Rate 2-way cache size N/2**
- **Beware: Execution time is only final measure!**
  - **Will Clock Cycle time increase?**
  - **Hill [1988] suggested hit time external cache +10%, internal + 2% for 2-way vs. 1-way**

Example: Avg. Memory Access Time vs. Miss Rate

- **Example: assume CCT = 1.10 for 2-way, 1.12 for 4-way, 1.14 for 8-way vs. CCT direct mapped**

Cache Size	Associativity			
(KB)	1-way	2-way	4-way	8-way
1	2.33	2.15	2.07	2.01
2	1.98	1.86	1.76	1.68
4	1.72	1.67	1.61	1.53
8	1.46	1.48	1.47	1.43
16	1.29	1.32	1.32	1.32
32	1.20	1.24	1.25	1.27
64	1.14	1.20	1.21	1.23
128	1.10	1.17	1.18	1.20

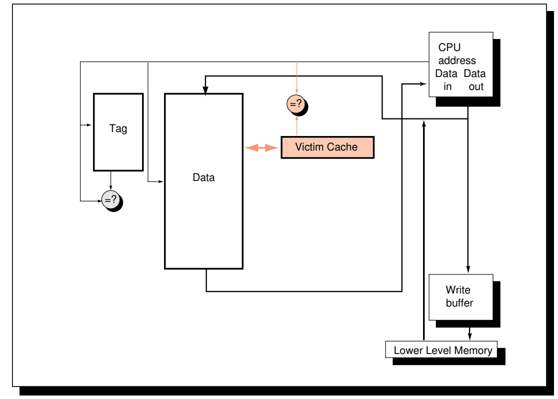
(Red means A.M.A.T. not improved by more associativity)

CS 252 Administrivia

- Sorry that last homework had high ratio of time to learning; next one will be better!
- Distribute Memory hierarchy homework, to be done in pairs
  - Due Monday Sept 30 by 5PM in box in 273 Soda
- Pick partners and tentative projects by Monday September 23? Send email Rich Fromm with partner, short description of topic
- Part of CS 252 is expose to architecture research projects underway at Berkeley
  - Wednesday Septemeber 25, guest lecture on Reconfigurable Computing, part of BRASS project just starting at Berkeley

3. Reducing Misses via Victim Cache

- How to combine fast hit time of Direct Mapped yet still avoid conflict misses?
- Add buffer to place data discarded from cache
- Jouppi [1990]: 4-entry victim cache removed 20% to 95% of conflicts for a 4 KB direct mapped data cache
- Used in Alpha, HP machines



4. Reducing Misses via Pseudo-Associativity

- How to combine fast hit time of Direct Mapped and have the lower conflict misses of 2-way SA cache?
- Divide cache: on a miss, check other half of cache to see if there, if so have a pseudo-hit (slow hit)

Hit Time

Pseudo Hit Time | Miss Penalty

Time

- Drawback: CPU pipeline is hard if hit takes 1 or 2 cycles
  - Better for caches not tied directly to processor (L2)

5. Reducing Misses by HW Prefetching of Instruction & Data

- E.g., Instruction Prefetching
  - Alpha 21064 fetches 2 blocks on a miss
  - Extra block placed in stream buffer
  - On miss check stream buffer
- Works with data blocks too:
  - Jouppi [1990] 1 data stream buffer got 25% misses from 4KB cache; 4 streams got 43%
  - Palacharla & Kessler [1994] for scientific programs for 8 streams got 50% to 70% of misses from 2 64KB, 4-way set associative caches
- Prefetching relies on extra memory bandwidth that can be used without penalty

6. Reducing Misses by SW Prefetching Data

- Data Prefetch
  - Load data into register (HP PA-RISC loads)
  - Cache Prefetch: load into cache (MIPS IV, PowerPC, SPARC v. 9)
  - Special prefetching instructions cannot cause faults; a form of speculative execution
- Issuing Prefetch Instructions takes time
  - Is cost of prefetch issues < savings in reduced misses?

7. Reducing Misses by Compiler Optimizations

- Instructions
  - Reorder procedures in memory so as to reduce conflict misses
  - Profiling to look at conflicts
  - McFarling [1989] reduced caches misses by 75% on 8KB direct mapped cache with 4 byte blocks
- Data
  - *Merging Arrays*: improve spatial locality by single array of compound elements vs. 2 arrays
  - *Loop Interchange*: change nesting of loops to access data in order stored in memory
  - *Loop Fusion*: Combine 2 independent loops that have same looping and some variables overlap
  - *Blocking*: Improve temporal locality by accessing “blocks” of data repeatedly vs. going down whole columns or rows

Merging Arrays Example

```
/* Before */
int val[SIZE];
int key[SIZE];

/* After */
struct merge {
  int val;
  int key;
};
struct merge merged_array[SIZE];
```

Reducing conflicts between val & key

Loop Interchange Example

```
/* Before */
for (k = 0; k < 100; k = k+1)
  for (j = 0; j < 100; j = j+1)
      for (i = 0; i < 5000; i = i+1)
             x[i][j] = 2 * x[i][j];
/* After */
for (k = 0; k < 100; k = k+1)
  for (i = 0; i < 5000; i = i+1)
      for (j = 0; j < 100; j = j+1)
             x[i][j] = 2 * x[i][j];
```

Sequential accesses Instead of striding through memory every 100 words

Loop Fusion Example

```
/* Before */
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
       a[i][j] = 1/b[i][j] * c[i][j];
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
       d[i][j] = a[i][j] + c[i][j];
/* After */
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
  {    a[i][j] = 1/b[i][j] * c[i][j];
       d[i][j] = a[i][j] + c[i][j];}
```

2 misses per access to a & c vs. one miss per access

Blocking Example

```
/* Before */
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
     {r = 0;
      for (k = 0; k < N; k = k+1){
        r = r + y[i][k]*z[k][j];};
      x[i][j] = r;
     };
```

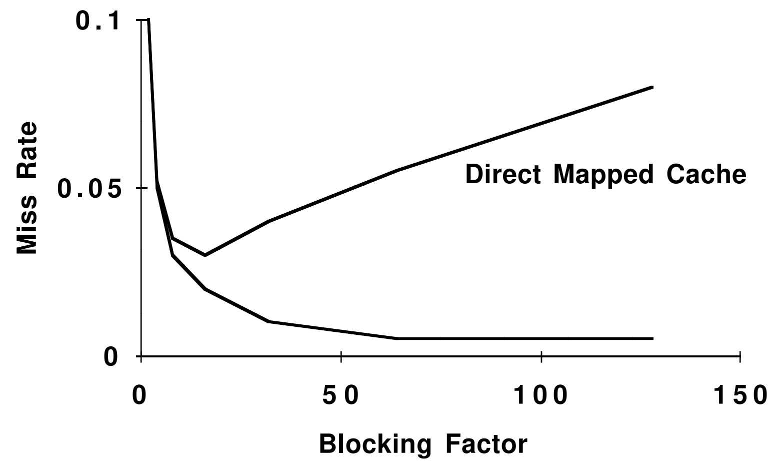
- Two Inner Loops:
  - Read all NxN elements of z[]
  - Read N elements of 1 row of y[] repeatedly
  - Write N elements of 1 row of x[]
- Capacity Misses a function of N & Cache Size:
  - 3 NxNx4 => no capacity misses; otherwise ...
- Idea: compute on BxB submatrix that fits

Blocking Example

```
/* After */
for (jj = 0; jj < N; jj = jj+B)
for (kk = 0; kk < N; kk = kk+B)
for (i = 0; i < N; i = i+1)
   for (j = jj; j < min(jj+B-1,N); j = j+1)
     {r = 0;
      for (k = kk; k < min(kk+B-1,N); k = k+1) {
        r = r + y[i][k]*z[k][j];};
      x[i][j] = x[i][j] + r;
     };
```

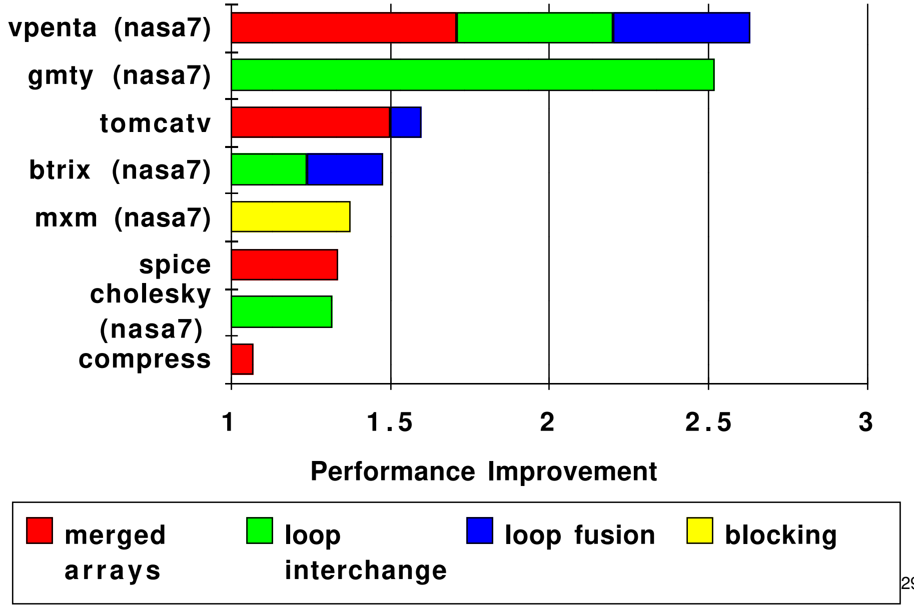
- Capacity Misses from $2N^3 + N^2$ to $2N^3/B + N^2$
- B called *Blocking Factor*
- Conflict Misses Too?

Reducing Conflict Misses by Blocking



- Conflict misses in caches not FA vs. Blocking size
  - Lam et al [1991] a blocking factor of 24 had a fifth the misses vs. 48 despite both fit in cache

Summary of Compiler Optimizations to Reduce Cache Misses



Summary

$$CPUtime = IC \times \left(CPI_{Execution} + \frac{Memory\ \ accesses}{Instruction} \times \textbf{Miss rate} \times Miss\ \ penalty \right) \times Clock\ \ cycle\ \ time$$

- 3 Cs: Compulsory, Capacity, Conflict Misses
- Reducing Miss Rate
  1. Reduce Misses via Larger Block Size
  2. Reduce Misses via Higher Associativity
  3. Reducing Misses via Victim Cache
  4. Reducing Misses via Pseudo-Associativity
  5. Reducing Misses by HW Prefetching Instr, Data
  6. Reducing Misses by SW Prefetching Data
  7. Reducing Misses by Compiler Optimizations
- Remember danger of concentrating on just one parameter when evaluating performance

5 minute Class Break

- Lecture Format:
  - ≈ 1 minute: review last time & motivate this lecture
  - ≈ 20 minute lecture
  - ≈ 3 minutes: discuss class manangement
  - ≈ 25 minutes: lecture
  - 5 minutes: break
  - ≈25 minutes: lecture
  - ≈1 minute: summary of today's important topics

Review: Improving Cache Performance

1. Reduce the miss rate,

2. Reduce the miss penalty, or

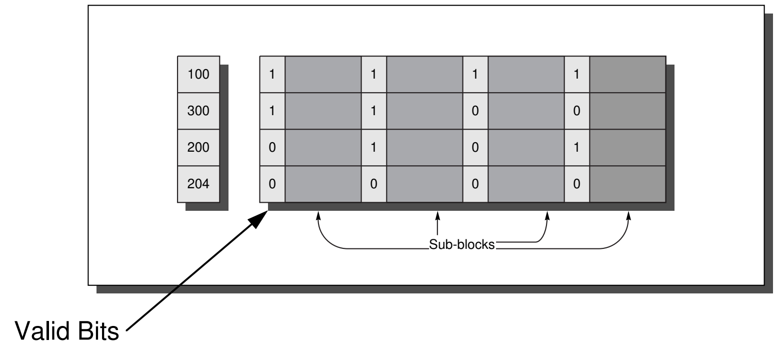
3. Reduce the time to hit in the cache.

1. Reducing Miss Penalty: Read Priority over Write on Miss

- Write through with write buffers offer RAW conflicts with main memory reads on cache misses
- If simply wait for write buffer to empty might increase read miss penalty by 50% (old MIPS 1000)
- Check write buffer contents before read; if no conflicts, let the memory access continue
- Write Back?
  - Read miss replacing dirty block
  - Normal: Write dirty block to memory, and then do the read
  - Instead copy the dirty block to a write buffer, then do the read, and then do the write
  - CPU stall less since restarts as soon as do read

2. Subblock Placement to Reduce Miss Penalty

- Don't have to load full block on a miss
- Have bits per subblock to indicate valid
- (Originally invented to reduce tag storage)



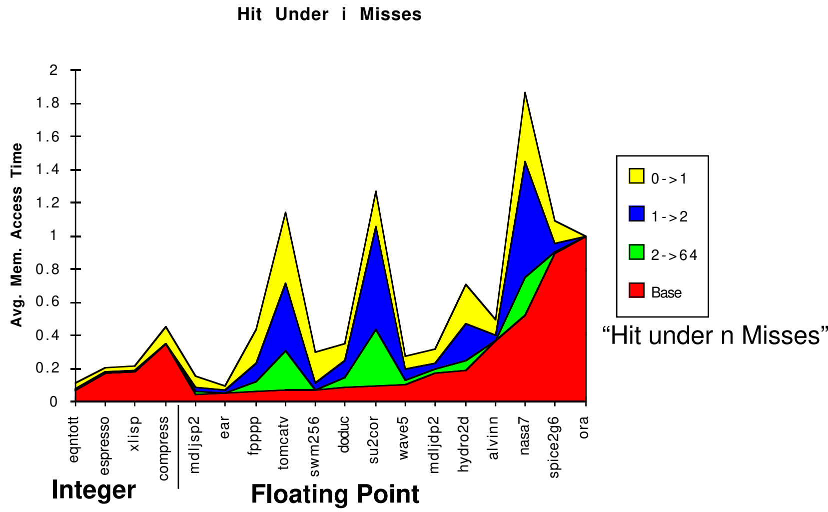
3. Early Restart and Critical Word First

- Don't wait for full block to be loaded before restarting CPU
  - *Early restart*—As soon as the requested word of the block ar rives, send it to the CPU and let the CPU continue execution
  - *Critical Word First*—Request the missed word first from memory and send it to the CPU as soon as it arrives; let the CPU continue execution while filling the rest of the words in the block. Also called *wrapped fetch* and *requested word first*
- Generally useful only in large blocks,
- Spatial locality a problem; tend to want next sequential word, so not clear if benefit by early restart

4. Non-blocking Caches to reduce stalls on misses

- *Non-blocking cache* or *lockup-free cache* allowing the data cache to continue to supply cache hits during a miss
- "*hit under miss*" reduces the effective miss penalty by being helpful during a miss instead of ignoring the requests of the CPU
- "*hit under multiple miss*" or "*miss under miss*" may further lower the effective miss penalty by overlapping multiple misses
  - Significantly increases the complexity of the cache controller as there can be multiple outstanding memory accesses

Value of Hit Under Miss for SPEC



- FP programs on average: AMAT= 0.68 -> 0.52 -> 0.34 -> 0.26
- Int programs on average: AMAT= 0.24 -> 0.20 -> 0.19 -> 0.19
- 8 KB Data Cache, Direct Mapped, 32B block, 16 cycle miss

5th Miss Penalty Reduction: Second Level Cache

- **L2 Equations**

  AMAT = Hit Time$_{L1}$ + Miss Rate$_{L1}$ x Miss Penalty$_{L1}$

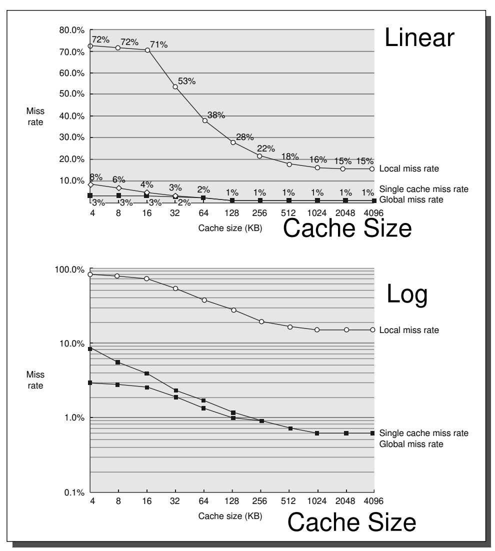
  Miss Penalty$_{L1}$ = Hit Time$_{L2}$ + Miss Rate$_{L2}$ x Miss Penalty$_{L2}$

  AMAT = Hit Time$_{L1}$ + Miss Rate$_{L1}$ x (Hit Time$_{L2}$ + Miss Rate$_{L2}$ + Miss Penalty$_{L2}$)

- **Definitions:**
  - *Local miss rate*— misses in this cache divided by the total number of memory accesses *to this cache* (Miss rate$_{L2}$)
  - *Global miss rate*—misses in this cache divided by the total number of memory accesses *generated by the CPU*
    (Miss Rate$_{L1}$ x Miss Rate$_{L2}$)

Comparing Local and Global Miss Rates

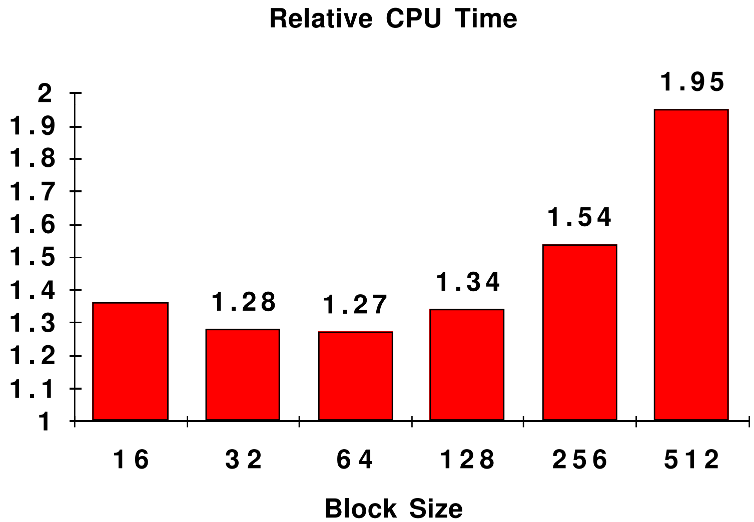
- 32 KByte 1st level cache; Increasing 2nd level cache
- Global miss rate close to single level cache rate provided L2 >> L1
- Don't use local miss rate
- L2 not tied to CPU clock cycle
- Cost & A.M.A.T.
- Generally Fast Hit Times and fewer misses
- Since hits are few, target miss reduction



Reducing Misses: Which apply to L2 Cache?

- **Reducing Miss Rate**
  1. Reduce Misses via Larger Block Size
  2. Reduce Conflict Misses via Higher Associativity
  3. Reducing Conflict Misses via Victim Cache
  4. Reducing Conflict Misses via Pseudo-Associativity
  5. Reducing Misses by HW Prefetching Instr, Data
  6. Reducing Misses by SW Prefetching Data
  7. Reducing Capacity/Conf. Misses by Compiler Optimizations

L2 cache block size & A.M.A.T.



- 32KB L1, 8 byte path to memory

Reducing Miss Penalty Summary

- Five techniques
  - Read priority over write on miss
  - Subblock placement
  - Early Restart and Critical Word First on miss
  - Non-blocking Caches (Hit Under Miss)
  - Second Level Cache
- Can be applied recursively to Multilevel Caches
  - Danger is that time to DRAM will grow with multiple levels in between

Review: Improving Cache Performance

1. Reduce the miss rate,
2. Reduce the miss penalty, or
3. *Reduce the time to hit in the cache*.

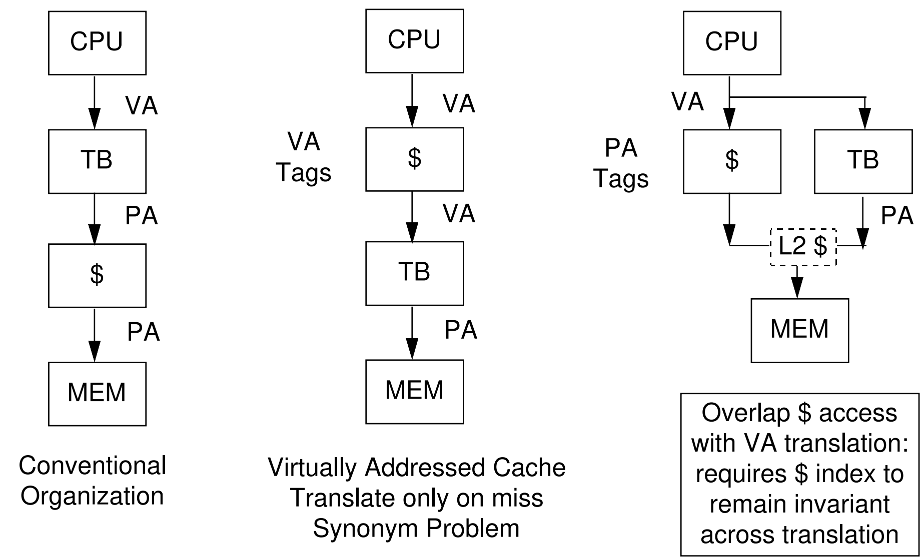
1. Fast Hit times via Small and Simple Caches

- Why Alpha 21164 has 8KB Instruction and 8KB data cache + 96KB second level cache
- Direct Mapped, on chip

2. Fast hits by Avoiding Address Translation

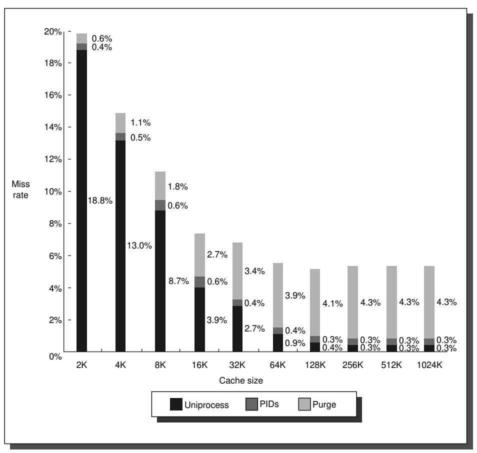
- **Send virtual address to cache? Called *Virtually Addressed Cache* or just *Virtual Cache* vs. *Physical Cache***
  - **Every time process is switched logically must flush the cache; otherwise get false hits**
    - » **Cost is time to flush + "compulsory" misses from empty cache**
  - **Dealing with *aliases* (sometimes called *synonyms*); Two different virtual addresses map to same physical address**
  - **I/O must interact with cache, so need virtual address**
- **Solution to aliases**
  - **HW that guarantees that every cache block has unique physical address**
  - **SW guarantee : lower n bits must have same address; as long as covers index field & direct mapped, they must be unique; called *page coloring***
- **Solution to cache flush**
  - **Add *process identifier tag* that identifies process as well as address**

Virtually Addressed Caches



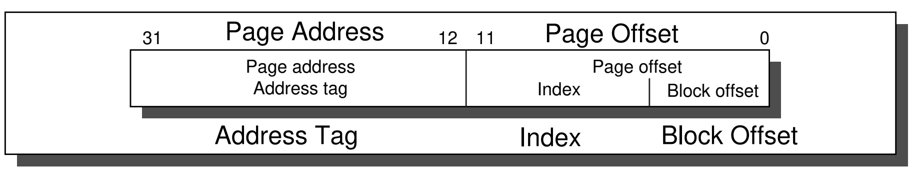
2. Avoiding Translation: Process ID impact

- **Black is uniprocess**
- **Light Gray is multiprocess when flush cache**
- **Dark Gray is multiprocess when use Process ID tag**
- **Y axis: Miss Rates up to 20%**
- **X axis: Cache size from 2 KB to 1024 KB**



2. Avoiding Translation: Index with Physical Portion of Address

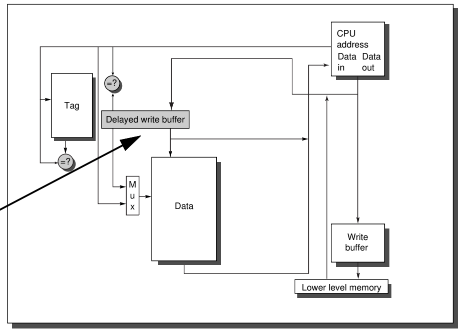
- **If index is physical part of address, can start tag access in parallel with translation so that can compare to physical tag**



- **Limits cache to page size: what if want bigger caches and uses same trick?**
  - **Higher associativity**
  - **Page coloring**

3. Fast Hit Times Via Pipelined Writes

- **Pipeline Tag Check and Update Cache as separate stages; current write tag check & previous write cache update**
- **Only Write in the pipeline; empty during a miss**



- **In color is Delayed Write Buffer; must be checked on reads; either complete write or read from buffer**

4. Fast Writes on Misses Via Small Subblocks

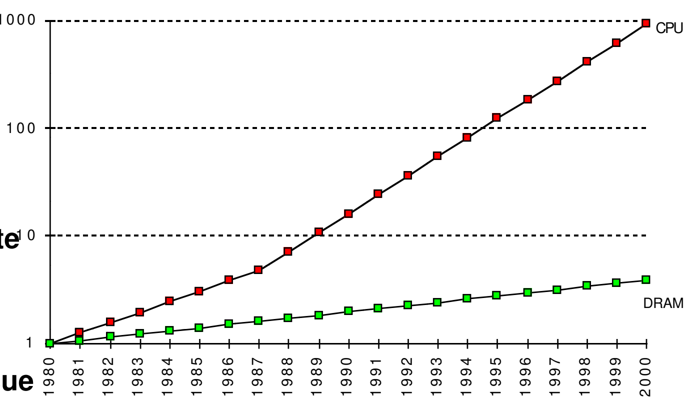
- **If most writes are 1 word, subblock size is 1 word, & write through then always write subblock & tag immediately**
  - ***Tag match and valid bit already set*: Writing the block was proper, & nothing lost by setting valid bit on again.**
  - ***Tag match and valid bit not set*: The tag match means that this is the proper block; writing the data into the subblock makes it appropriate to turn the valid bit on.**
  - ***Tag mismatch*: This is a miss and will modify the data portion of the block. As this is a write-through cache, however, no harm was done; memory still has an up-to-date copy of the old value. Only the tag to the address of the write and the valid bits of the other subblock need be changed because the valid bit for this subblock has already been set**
- **Doesn't work with write back due to last case**

Cache Optimization Summary

Technique	*MR*	*MP*	*HT*	*Complexity*
Larger Block Size	+	–		0
Higher Associativity	+		–	1
Victim Caches	+			2
Pseudo-Associative Caches	+			2
HW Prefetching of Instr/Data	+			2
Compiler Controlled Prefetching	+			3
Compiler Reduce Misses	+			0
Priority to Read Misses		+		1
Subblock Placement		+	+	1
Early Restart & Critical Word 1st		+		2
Non-Blocking Caches		+		3
Second Level Caches		+		2
Small & Simple Caches	–		+	0
Avoiding Address Translation			+	2
Pipelining Writes			+	1

What is the Impact of What You've Learned About Caches?

- **1960-1985: Speed = ƒ(no. operations)**
- **1995**
  - **Pipelined Execution & Fast Clock Rate**
  - **Out-of-Order completion**
  - **Superscalar Instruction Issue**



- **1995: Speed = ƒ(non-cached memory accesses)**
- **What does this mean for**
  - **Compilers?,Operating Systems?, Algorithms? Data Structures?**