Review: Summary

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

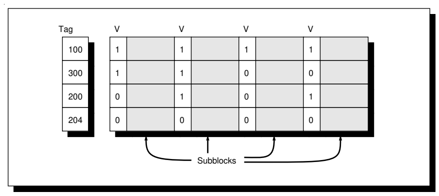
- **3 Cs: Compulsory, Capacity, Conflict Misses**
- **Reducing Miss Rate**
  1. **Reduce Misses via Larger Block Size**
  2. **Reduce Conflict Misses via Higher Associativity**
  3. **Reducing Conflict Misses via Victim Cache**
  4. **Reducing Conflict Misses via Pseudo-Associativity**
  5. **Reducing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

1. Reducing Miss Penalty: Read priority over write on miss

- **Write back with Write buffers offer RAW conflicts with main memory reads on cache misses**
- **If simply wait for write buffer to empty might increase read misses by 50% (old MIPS 1000)**
- **Check write buffer contents before read; if no conflicts, let the memory continue**
- **Write Back?**
  - **read miss replacing dirty block**
  - **Normal: Write dirty block to memory, and then doing 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)



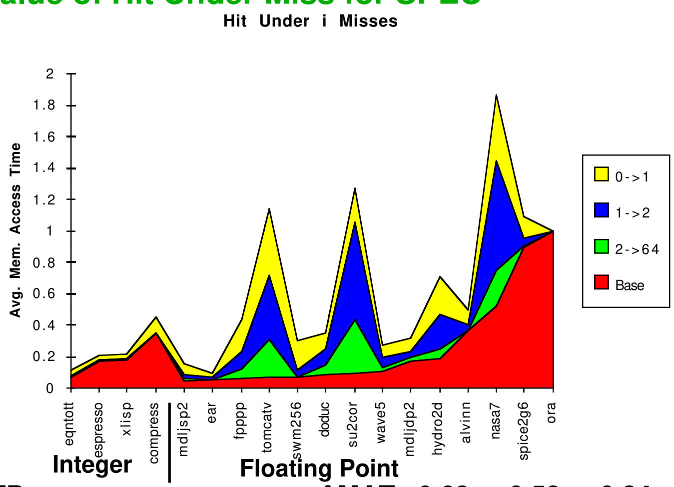
3. Early Restart and Critical Word First

- 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,
- Spacial 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 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**

$$Average\ \ memory\ \ access\ \ time = Hit\ \ time_{L1} + Miss\ \ rate_{L1} \times Miss\ \ penalty_{L1}$$

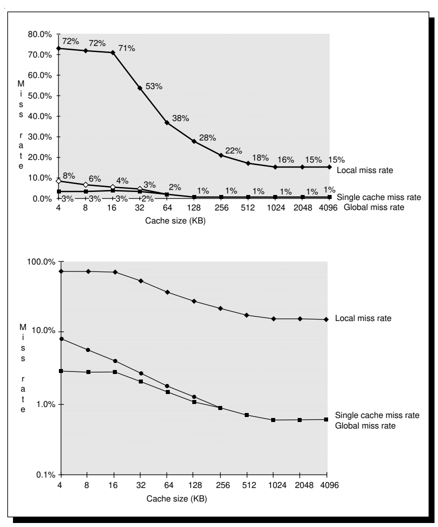
$$Miss\ \ penalty_{L1} = Hit\ \ time_{L2} + Miss\ \ rate_{L2} \times Miss\ \ penalty_{L2}$$

$$Average\ \ memory\ \ access\ \ time = Hit\ \ time_{L1} + Miss\ \ rate_{L1} \times \left(Hit\ \ time_{L2} + Miss\ \ rate_{L2} \times Miss\ \ penalty_{L2}\right)$$

- **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

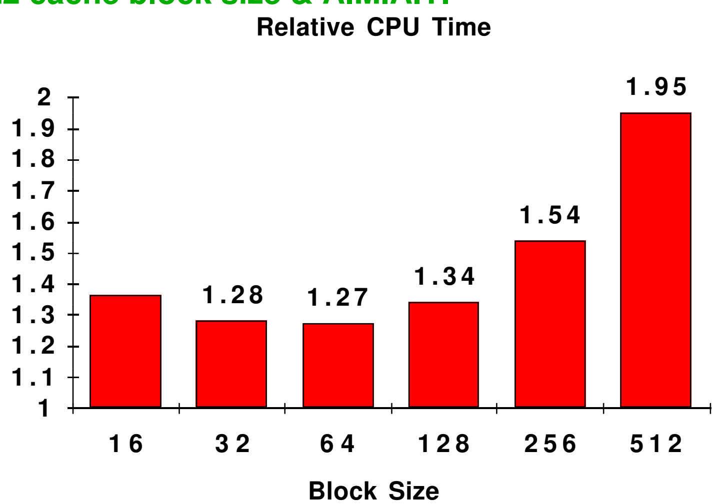
- **Linear vs. Log scale**
- **Global miss rate close to single level cache rate provided L2 >> L1**
- **Don't use local miss rate**
- **L2 not tied to CPU clock cycle**
- **Cost & A.M.A.T.**
- **Generally Fast Hit Times and fewer misses**
- **Since hits are few, target miss reduction**



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

- 5 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

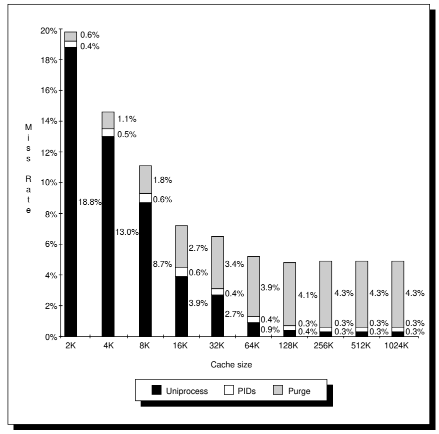
- **Why Alpha 21164 has 8KB Instruction and 8KB data cache + 96KB second level cache**
- **Direct Mapped, on chip**

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*); 2 different virtual address 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 within process: can't get a hit if wrong process**

2. Avoiding Translation: Process ID impact

- Black is uniprocess
- Gray is multiprocess when flush cache
- White 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

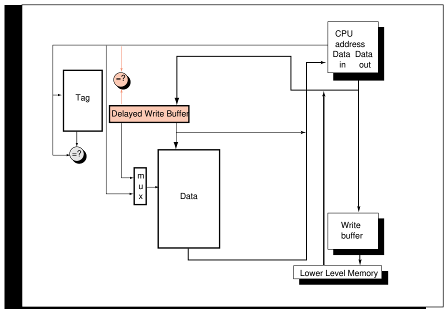
- If index is physical part of address, can start tag access in parallel with translation so that can compare to physical tag

31 12	11	0
Page Address	Page offset	
Address Tag	Index	Block offset

- 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 tag 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

Impact of What You've Learned About Caches?

- 1960-1985: Speed = f(no. operations)
- 1995
  - Pipelined Execution & Fast Clock Rate
  - Out-of-Order completion
  - Superscalar Instruction Issue
- 1995: Speed = f(non-cached memory accesses)
- What does this mean to
  - Compilers?
  - Operating Systems?
  - Algorithms?
  - Data Structures?