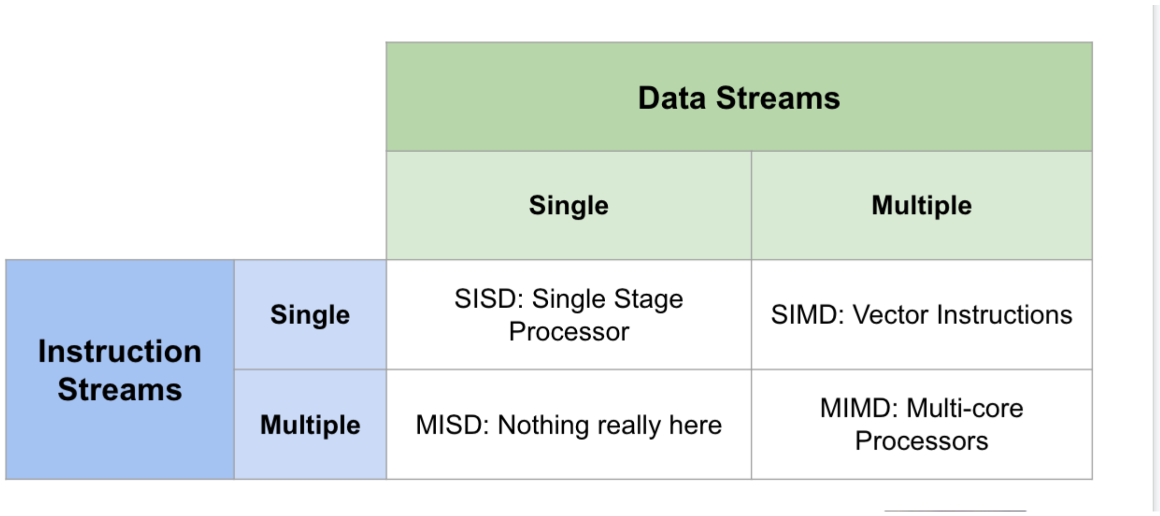
Performance, DLP

- Moore's Law : Every two years, the number of transistors on a chip of a fixed size doubles $\rightarrow$ stopped $\sim$ 2006
- Domain-specific Hardware & data-level Parallelism (SIMD + unrolling)
- Flynn's Taxonomy



		Data Streams	
		Single	**Multiple**
Instruction Streams	**Single**	SISD: Single Stage Processor	SIMD: Vector Instructions
	Multiple	MISD: Nothing really here	MIMD: Multi-core Processors

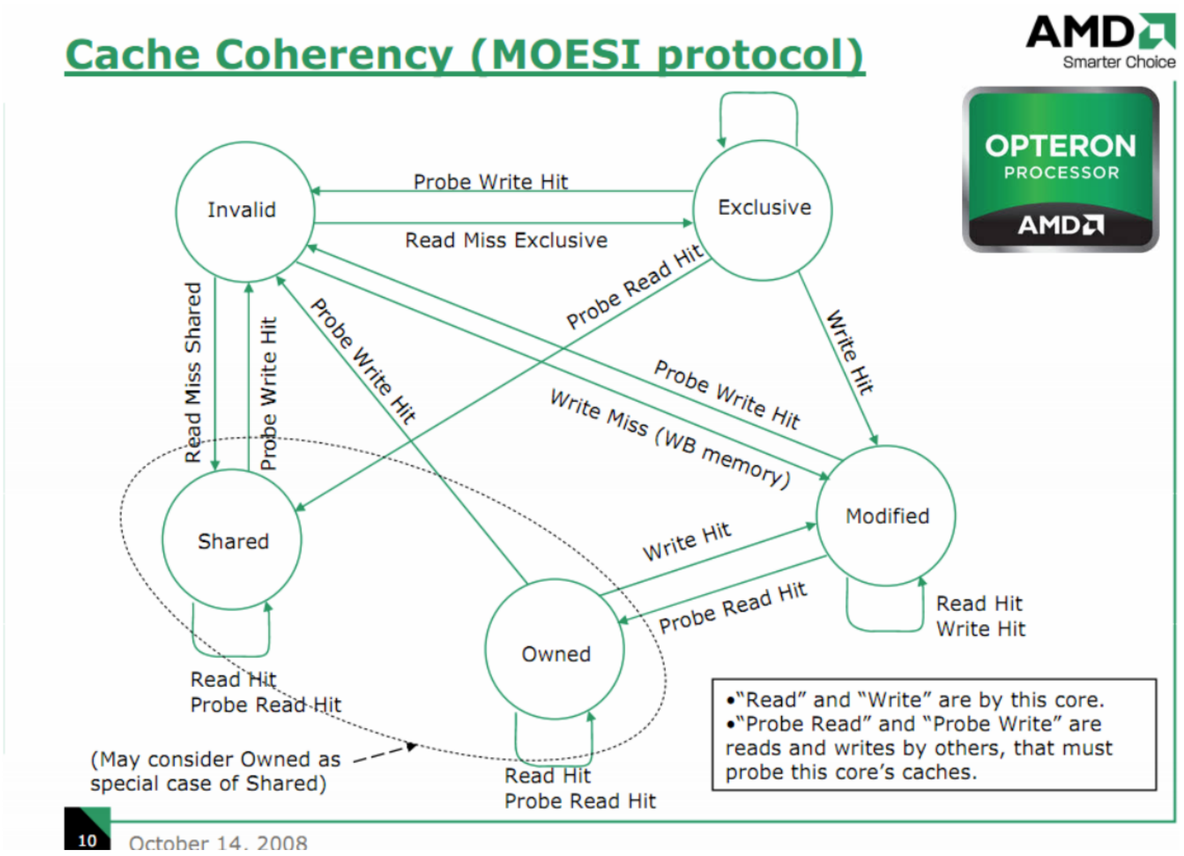
- SIMD : data-level parallelism (DLP); Note : casting when necessary, care about ptr and actual data
- Loop unrolling : main and tail case
- Amdahl's Law : true speedup $= \frac{1}{S+\frac{1-S}{P}} = \frac{P}{PS+1-S}$ where S is the serial part and P is the speedup factor

OMP

- Vars declared outside of omp parallel is shared, declared inside is private
- Each thread has their own version of stack memory, but NOT their own code or static memory
- #pragma omp parallel : needs {} around code; must be on individual lines
- A #pragma omp (parallel) for section : no extra curly braces. Just stick that directive directly above a for loop; breaks index in chunks (the for-loop index is always private)
- Pitfall : can reduce speedup or break program logic
  - Data Dependencies, e.g. Fibonacci
  - Sharing, e.g. temp in swap
  - (Data Race) Updating Shared Variables Simultaneously, e.g. sum
  - Parallel Overhead → Better to have fewer but larger parallel regions
  - False Sharing (diff threads editing diff bytes in same block), e.g. parallel for with small chunk
- Cache Coherence : each core has a local private cache
  - Every processor store instruction must check the contents of all other caches
  - Coherence Protocol : MOESI
    - You cannot write to a Shared state (!) Shared means multiple caches have a copy. In order to write to a location multiple caches have, you have to first "kick out" or Invalidate those other states. This makes you the Exclusive holder. Then, you can transition from Exclusive to Modified by writing.
    - You can "escape" a Shared state by being invalidated yourself, or by invalidating the other caches which share your data.
    - If you want to dirty something in a Shared state (ie make it modified) you have to Invalidate your sharers, transition to Exclusive, then do your write and move to Modified.
    - If in the Exclusive state, a processor can write without notifying other caches
    - Owner state is a variation of Shared state to let caches supply data instead of going to memory on a read miss
    - Exclusive state is a variation of Modified state to let caches avoid writing to memory on a miss

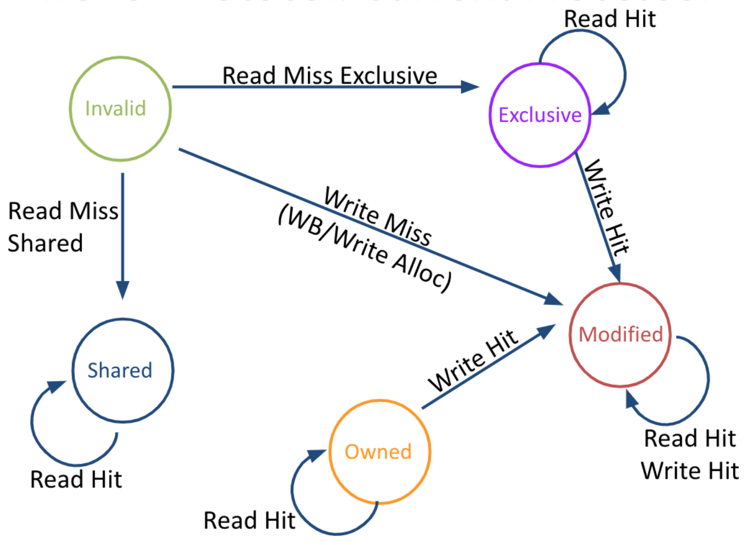
- Each block in each cache is in one of the following states:
  - Modified (in cache)
  - Owned (in cache)
  - Exclusive (in cache)
  - Shared (in cache)
  - Invalid (not in cache)

	M	O	E	S	I
M	X	X	X	X	✓
O	X	X	X	✓	✓
E	X	X	X	X	✓
S	X	✓	X	✓	✓
I	✓	✓	✓	✓	✓



- "Read" and "Write" are by this core.
- "Probe Read" and "Probe Write" are reads and writes by others, that must probe this core's caches.

MOESI Protocol: Current Processor



	Valid Bit	Dirty Bit	Shared Bit
Modified	1	1	0
Owned	1	1	1
Exclusive	1	0	0
Shared	1	0	1
Invalid	0	X	X

- False Sharing
  - Block ping-pongs between two caches even though processors are accessing disjoint variables
  - Prevent by placing same-block data on different threads OR reduce block size
  - If same piece of data being used by 2 caches, ping-ponging is inevitable (NOT false sharing), i.e. Would cache invalidation occur if block size was only 1 word?
    Yes: true sharing; No: false sharing

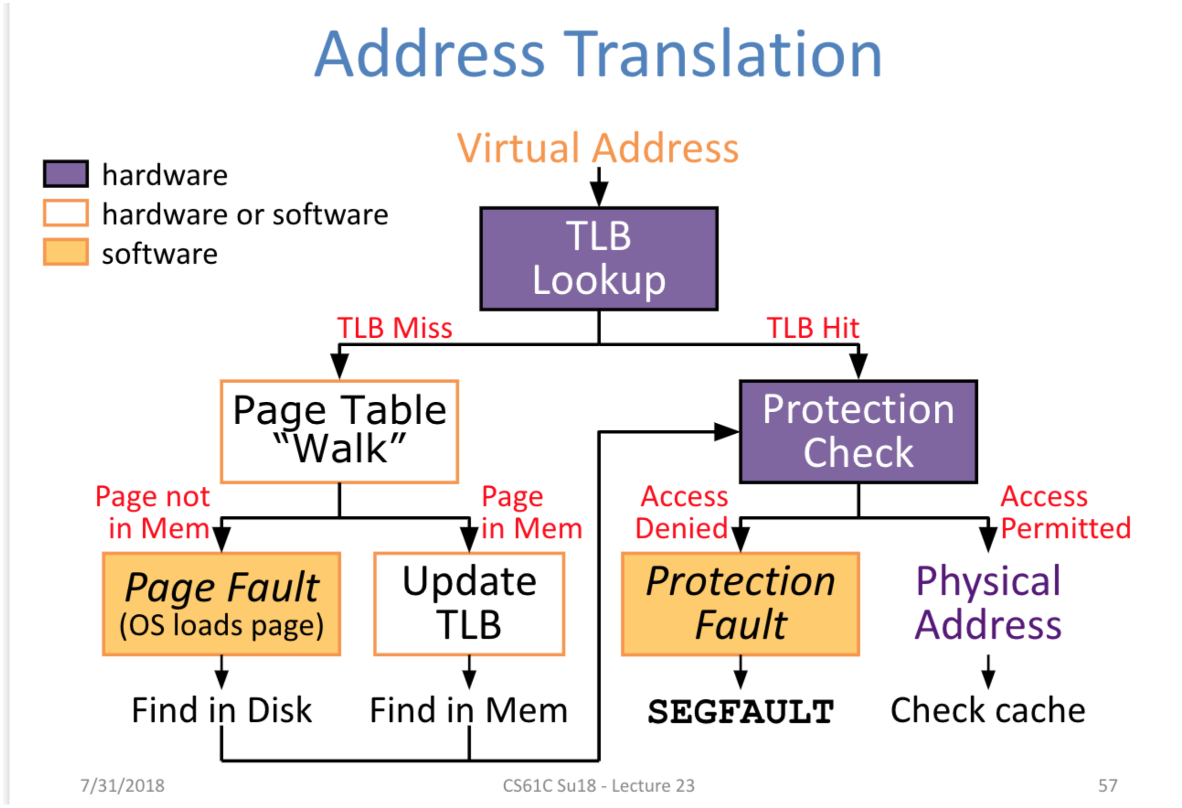
WSC, Cloud, RLP & MapReduce, Spark

- WSC : warehouse scale computers
  - Cost-efficiency (due to scale)
  - Almost half power goes to cooling (+ distribution)
  - Uses almost half peak power even completely in idle
  - Very highly available ($<$ 1 hr down/yr), even though lots of failures
  - Server ($\sim$ a disk) $\rightarrow$ rack $\rightarrow$ array
  - Workload variation
  - Goal : Energy-Proportionality, i.e. % peak load = % peak energy
  - PUE (power usage effectiveness) : $\frac{\text{total power}}{ITpower}$, 1.0 lowest, lower = better
- Cloud Computing
  - 5-7x cheaper than a medium-size (1000 servers) facility
  - Shared platform w/ illusion of isolation, low-cost cycles, elastic service (pay as you use)
  - SaaS (software as a service) : communicate with internet, e.g. Google Docs
  - Paas (platform) : e.g. Hadoop, Apache Spark
  - IaaS (infrastructure) : buy resource, e.g. Google Computer Platform, AWS
- RLP
  - High request volume, each largely independent
  - Replication for better throughput & availability
  - RLP uses more reads than writes
  - RLP runs naturally independent requests in parallel
  - RLP also runs independent tasks within a request
  - Make copies & load balance
    Search uses redundant copies of indices and data to deliver parallelism + prevent failures
  - Break up hot spots
- MapReduce
  - Convenient DLP on large dataset across large # of machines
  - Specify a map() and a reduce(), use emit to yield an Iterable
    map(in_key, in_val): ... emit(interm_key, interm_val), i.e. slice data into pieces;
    reduce(interm_key, list(interm_val)) : ... emit(out_key, out_val), i.e. combines all intermediate values for a particular key
  - Hides details of parallelism, fault tolerance, locality optimization, and load balancing
  - Divides computers into 1 master and N-1 workers; master assigns; Towards the end, the master assigns uncompleted tasks again; 1st to finish wins
  - Reduce work sorts by intermediate keys to group all occurrences of the same key
  - There are typically many more Map Tasks than Reduce Tasks
  - False : Reducers can start reducing as soon as they start to receive Map data
    True : Reduce begins as soon as all data sort finish, data sort begins as soon as a given Map finishes

- Spark : a MapReduce framework
  - Word count : file.flatMap(lambda line : line.split())
    .map(lambda word : (word, 1))
    .reduceByKey(lambda a, b : a + b)

Virtual Memory (VM)

- PM : physical memory; physical address bits = $\log_2$(PM space); vitual address bits = $\log_2$(VM space)
- PT : page table (in PM)
- Page offset : bits = $\log_2$(page size) ! same for both
- VPN (virtual page #) : bits = virtual address bits - page offset bits
- PPN (physical page #) : bits = physical address bits - page offset bits
- PTBR (page table base register) : address of the PT in physical memory (for any process), bits = $\log_2$(physical memory size)
- TLB Reach : amount of virtual address space that can be simultaneously mapped by TLB, size = # entries $\times$ page size
- Linear PT
  - Large & sparsely populated
  - Two accesses of PM for each process : 1 PTBR to PT, 2 PT to correct physical address
- Hierarchical PT
  - Have a PT (level 1) for all the PT's (level 2)
  - Use less space (each PT is smaller), but slower (three accesses of PM)
- TLB : Cache for VM, stores not data, but VPN $\rightarrow$ PPN mappings, i.e. VPN input with PPN output, on miss access PT in PM
  - Valid, Ref, Access Rights, VPN, PPN
- Fetch Data on Mem Read
  1. Check TLB (input : VPN, output PPN)
     - TLB Hit : fetch translation, return PPN
     - TLB Miss : check PT (in PM)
       * PT Hit :
       * PT Miss (Page Faults) ...
  2. Check cache (input : PPN, output : data)
- Protection Fault : the page table entry for a virtual page has permission bits that prohibit the requested operation.
- Page Fault : the page table for a virtual page has valid bit = 0 (false), i.e. entry not in memory
- Benefit : 1. illusion of infinite mem; 2. simulates full address space for each process so linker/loader don't care about other programs; 3. enforces protection b/w processes & w/i (e.g. read-only pages)



Address Translation

I/O

- For less frequent (mouse, keyboard), do Polling; for high frequency (hard drives, network cards), do Interrupt
- DMA (direct memory access) most beneficial for transferring large data blocks
- Very infrequent data : interrupt

Dependability

- Reliability : Mean Time To Failure (MTTF)
- Service interruption : Mean Time To Repair (MTTR)
- Mean Time Between Failures, MTBF = MTTR + MTTF
- Availability = $\frac{MTTF}{MTTF+MTTR} = \frac{MTTF}{MTBF}$
- Hamming ECC : parity bit set so that xor into 0 always
- RAID (redundant array of inexpensive disks, invented for performance)
  - All improve performance, but not all (0) improve reliability
  - RAID 0 : bit stripping, copy of data = 1; MTTF for single disk = T & k disks, then MTTF for array is T/k
  - (Insert chart in Disc 14)
  - RIAD 1
  - RAID 2 : bit stipping

Extra Sanity Checks

- Cache : a[i] += a[j] is 3 accesses, 2 reads 1 write
- Double-check if cache-aligned or page-aligned is given
- N.B. Use casting for SIMD
- N.B. page might not be allowed to write to (going to protection fault)
- Do ptr casting for malloc/calloc/realloc
- Pseudo : j, jr, ret; non-pseudo : jal, jalr; ret = jr ra = jalr x0, ra, 0
- The “power wall” meant that faster speed could no longer be achieved via higher clock rates and higher power per chip