THE UNIVERSITY of York

High Performance Computing - History and Internals of a Typical Computer

Prof Matt Probert

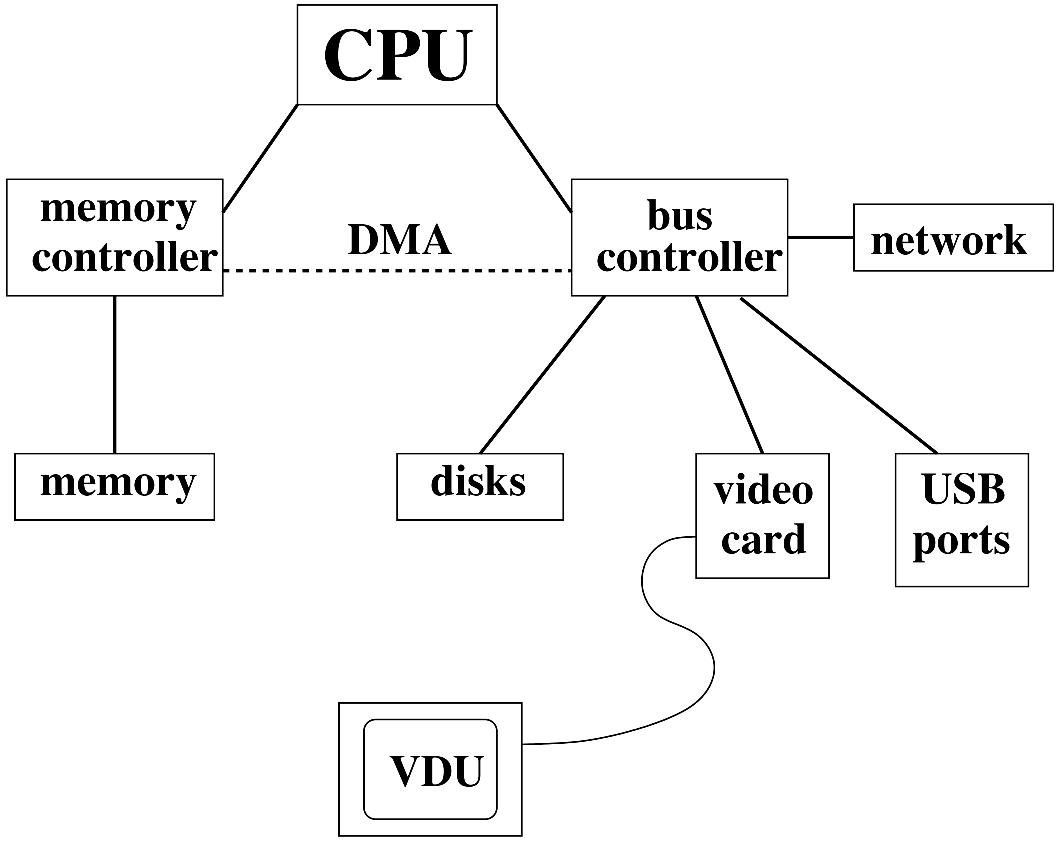
http://www-users.york.ac.uk/~mijp1

Overview

- How does a single-core computer work?
- What are the key subsystems that affect performance?
- Where are the computational bottlenecks?
- What are caches and pipelines?
- How does the computer ensure data integrity?

Traditional Single CPU Computer

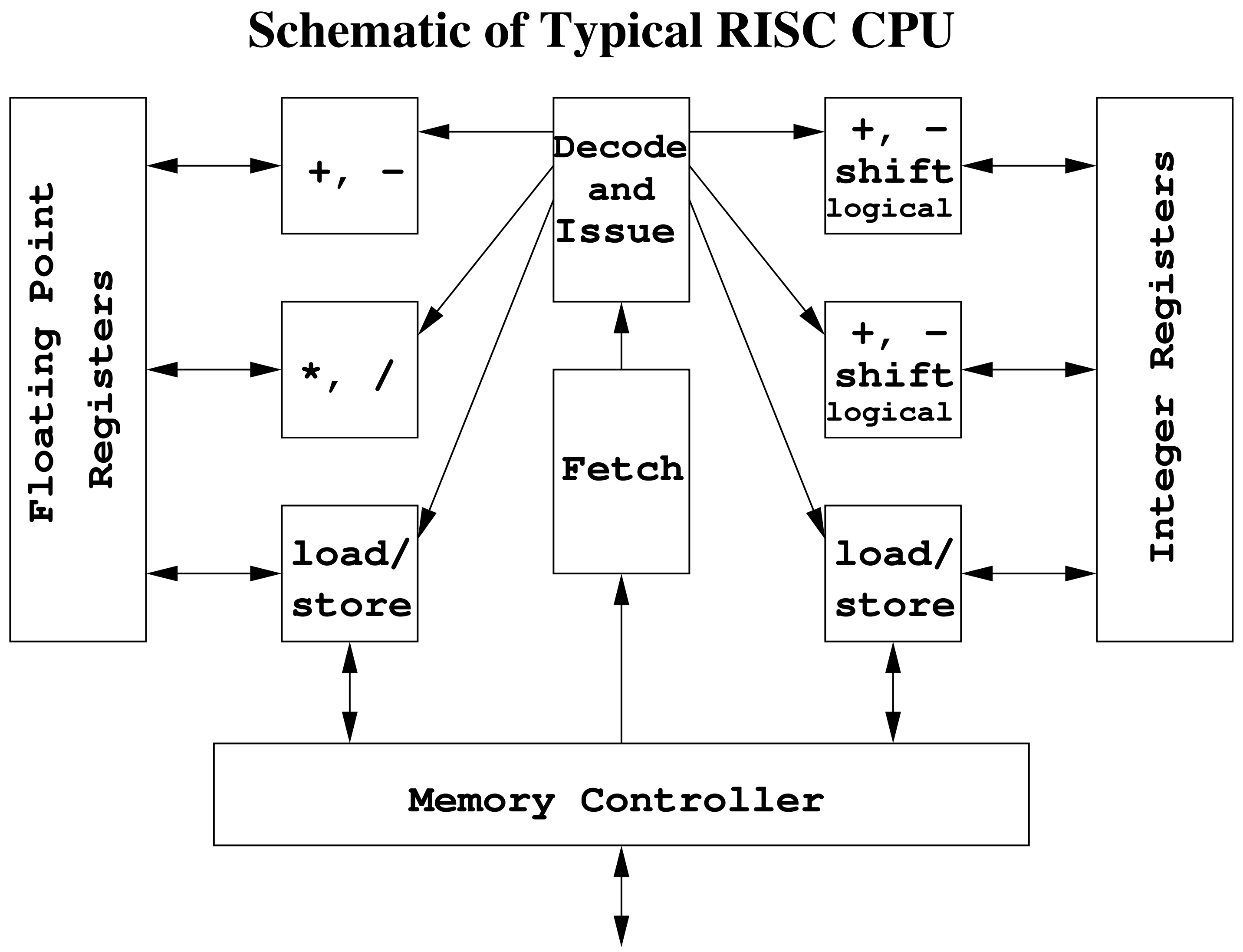
- Central Processing Unit communicates to other subsystems via a *bus*
- Key subsystems include memory, network, disks and video



Central Processing Unit (CPU)

- Understands two types of data: integer and floating point
  - may understand variants such as different precisions
  - Performs basic arithmetic operations and comparisons (can also be used for branches)
- Only understands machine code
  - each family of processors has completely different variant of machine code.

Typical RISC CPU



Parts of a CPU

- *Instruction fetcher* gets next machine code instruction from the *memory controller* and passes it on to the
- *Instruction decoder* which decodes instruction, and sends relevant data on to the
- *Functional units* which are dedicated to performing single operation (e.g. +,* etc) which use the
- *Registers* to store the input and output of the functional units (typically 32 floating point registers, and 32 integer registers)
  – Floating point and integer operations often handled in separate parts of CPU. In some older CPUs (e.g. Intel 80286 etc) there was no hardware support for FP – had a separate "numeric co-processor" (80287) instead!
- NB Memory (except cache – see later) is not part of the CPU and is used to store both program and data.

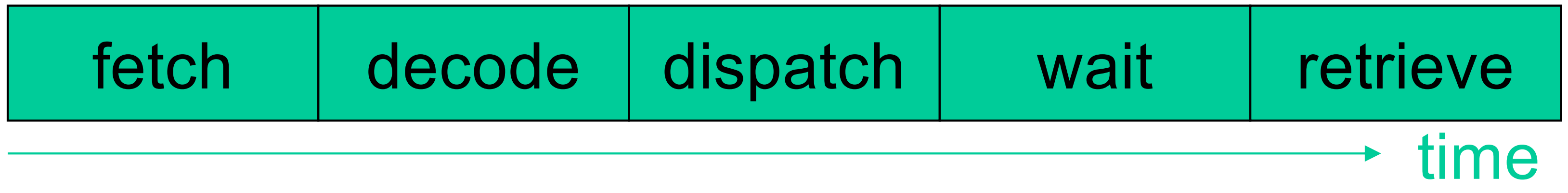
CPU Clock

- The *clock* is often (wrongly) used as a measure of the speed of a computer
  – Different CPU families do different amounts of work per clock cycle so cannot use this with different CPU designs
- The clock is simply an external signal (square pulses) used to synchronise parts of the system
- *Bus* is used to move data between subsystems – characterised by *width* (number of bits in parallel) as well as speed – internal usually much faster than external
  – There are buses within the CPU, and buses between different components on the motherboard (e.g. PCI etc), and external buses to peripherals (e.g. USB, SCSI, etc)

CPU Pipelines

- To execute a single machine code instruction requires several events in sequence

e.g.



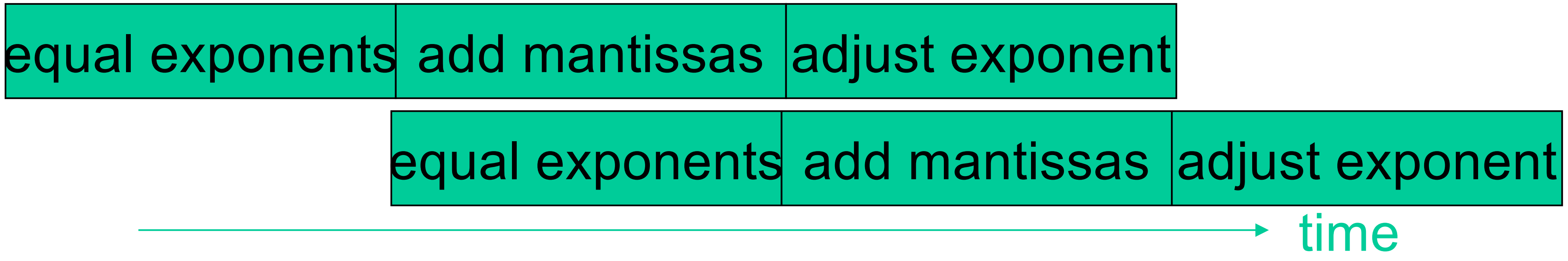
- There is a *pipeline* of operations that have to be performed sequentially
  - Pipeline has 5-25 *stages* depending on CPU design
  - So if each pipeline stage takes a single clock-cycle to complete then the above instruction would take 5 cycles to complete
  - But if we have multiple pipelines, then one part of the CPU can be fetching whilst another part is decoding, etc so can overlap independent operations (as long as no branching)

RISC or CISC?

- Can also overlap arithmetic operations if no data dependency in arguments
- Can also have multiple instruction decoders – a.k.a. *superscalar* CPUs
  - Enables instruction-level parallelism using spare functional units but decoding can be complicated unless all instructions same length (e.g. 4 words)
  - And scheduling overlapping operations is complex if different instructions vary wildly in execution time
- Is it better to have a small number of simple instructions, or a large set of complex ones?
- Early CPUs were CISC, modern ones are "RISC" but instruction sets have crept up in size and so distinction now less clear!

Overlapping Arithmetic

- Can also have pipelines within each functional unit
- Consider floating-point addition in base10:
  - $1.23*10^5+4.0*10^4$: first we need to adjust exponents to be equal ($12.3*10^4+4.0*10^4$), then add the mantissas ($12.3+4.0=16.3$), then adjust if necessary the exponents ($1.63*10^5$)
  - i.e. several distinct operations!
- So FP-add takes at least 3 clock cycles but with 3 stage pipeline these distinct operations can be overlapped, so FP-add can have a *latency* of 3 clock cycles but a *repeat rate* of 1 clock cycle.



Compiler Actions

- The compiler for a given high-level language needs to understand the underlying hardware (e.g. number and depth of pipelines) to get optimal performance

E.g. $\sum_{i=1}^{n} a_i$ looks like min. 3 cycles per term but a smart compiler would change

```
do i=1,n
   sum=sum+a(i)
end do
```

into

```
do i=1,n,3
   s1=s1+a(i)
   s2=s2+a(i+1)
   s3=s3+a(i+2)
end do
```

So no data dependency
and hence can issue 1 add/cycle!

Branching and Speculation

- So to get the best efficiency you need to keep the pipelines as full as possible and hence minimize the impact of latency
- So what if there is a branch in the code?
  - Processor cannot load the instructions following the branch instruction and hence the pipeline will *stall*.
- *Speculation* can improve this
  - If the CPU can *predict* the result of a conditional branch *before* the calculation is made, then it can pre-load the instructions following the branch whilst waiting for the branch itself to be evaluated!
  - If branch is then correctly predicted there will be no loss of efficiency. Otherwise the speculative results have to be discarded and pipeline is emptied

Out-of-Order Execution

- A further optimisation that can be achieved by more sophisticated CPU hardware is out-of-order execution
  - e.g. if an instruction is waiting for an earlier calculation to be completed due to a data dependency, then a later instruction with no dependencies can be scheduled in its place!

Threading

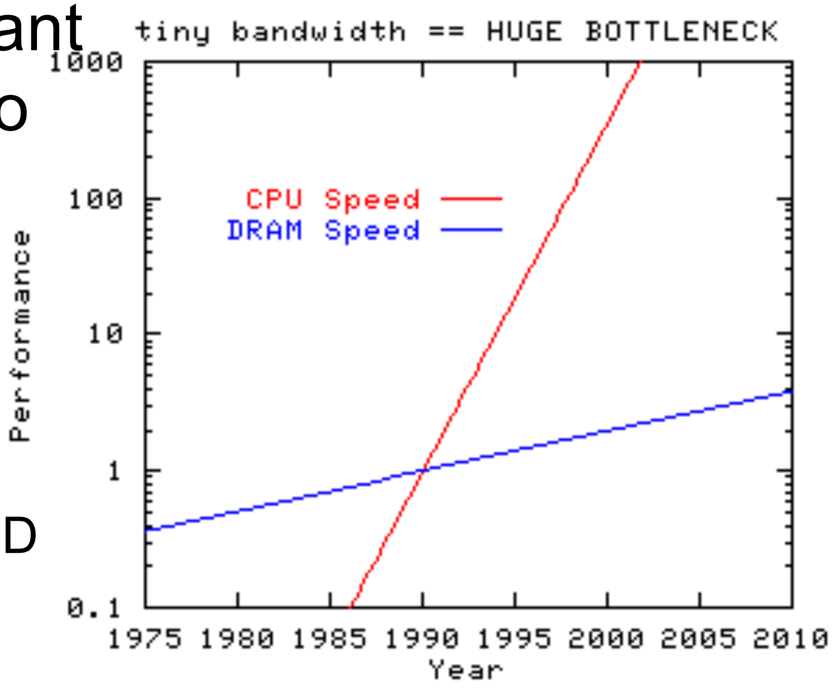
- A generic concept that can be implemented by O/S (multi-tasking) or within the CPU
  - CPU version has advantage of much faster process-switching time
- Called "Hyper-Threading" by Intel in 2002 with 2 threads/processor, but first realised by Tera (now Cray) in 1991 in its MTA with 128 threads/proc!
  - Each CPU thread has its own bank of registers but shares access to functional units and caches
  - Thread can execute whilst other is stalled, etc.
  - Extra logic in CPU gives ~10% increase in size but performance advantage can be considerable (~25%)
  - Each CPU appears to O/S as multiple (virtual) processors and can run totally distinct processes
  - Increased power dissipation and problems with caches

Multi-Core

- Logical next step to multi-threaded CPU
- Double up everything, not just the registers
  - Advantage is increased computational power without the need for higher clock frequencies which are becoming increasingly complex (feature size) and hot!
- IBM Power4 the first in 2001
  - Intel and AMD did first dual-cores in 2005
  - Then quad-core in 2006, hex-core in 2010, AMD octo-core in 2011, Intel in 2014, etc.
- Can also be combined with threading for even more virtual processors!
  - AMD EPYC3 (Milan) in March 2021 is 64-core with hyperthreading so 128 threads per CPU - $5k+ each …

Memory

- Memory is a key component of computer
- Often a computational bottleneck!
  - The problem is that as CPUs have got faster, memory has failed to keep up
  - From 1984 to 2004 CPU clock speed went from 4 MHz to 4 GHz* but memory to only 400 MHz
  - NB capacity has grown dramatically – from 64 kB to 512 GB DDR5 memory modules! And now 3600 MHz.
  - Such details might not be relevant to an MS Word user but are key to HPC performance!



* In Nov 2004 Intel changed its approach to faster PCs and shelved its plans for a 4 GHz Pentium 4 CPU - switched instead to multi-core approach. AMD then followed suit. No normal 4+ GHz chips …

The Memory Problem

- A typical CPU runs at 2.0 GHz. With multiple pipes, it might perform a double precision (8 byte) add and a multiply each clock-cycle => 4 data items in, 2 out
  - Memory controller needs a latency of 0.5/6 = 0.083 ns to deliver this (say 0.5 ns if latencies can be overlapped) and a bandwidth of 6 items x8 bytes x2 GHz = 96 GB/s
  - DDR2-1066 DDRAM has a 266 MHz clock with 64 bits output on leading & trailing edge with clock doubling (hence 1066 million transfers/second) with latency ~ 4 ns and peak bandwidth of 8.5 GB/s
  - DDR3 released in 2007 with 2* bandwidth but worse latency! DDR4 in 2014, DDR5 in 2020 – max 51.2 GB/s …
  - Practical figures worse due to controller chip latencies – hence AMD "HyperTransport" integrates controller + CPU
  - **Serious mismatch between memory and CPU**

Improving the Memory Bus

- Wider buses
  - 1st PCs has 16-bit memory buses, 386 & 486 had 32-bit, early Pentium had 64-bit
  - Now modern PC has 128-bit if “dual channel”
  - Alpha, Sun, SGI workstations up to 256-bit
  - But wide buses need many tracks on motherboard and lots of pins on CPU – cannot keep scaling this indefinitely
    - Intel LGA-1200 socket has 1200 contacts!
- And it does not address the latency problem …

Solving the Memory Bottleneck

- Use *caches* to form a *memory hierarchy*
  - Use a small amount of fast & expensive memory to store data which is frequently accessed
  - Uses a different, much more expensive hardware technology – SRAM requires 6 transistors/bit c.f. DRAM requires only 1
  - Cache *hit rate* is the number of memory accesses supplied by cache / total number of memory accesses requested – want it to be near 100% ideally
  - Requires very fast (and simple) control logic in *cache controller* so as not to slow things down
  - Many different policies, e.g. direct-mapped cache, set-associative cache – see wikipedia article for details

Memory Hierarchy

- Primary cache (L1)
  - Small (16 kB to 128 kB) on CPU, fast as possible
  - Takes 1-3 clock cycles to serve a memory request
- Secondary cache (L2)
  - Larger (256 kB to 8 MB), separate chip on motherboard or more recently integrated into CPU package
  - May be shared between cores
  - Takes 5-25 clock cycles to serve a memory request
- Sometimes
  - L3 cache – 2MB-64 MB shared between cores
- Main Memory (1 GB+)
  - Takes 30-300 clock cycles to serve a memory request
- Hard Disk (1000 GB+) – can use as extra memory via paging or swap files but *much* slower

How Reliable is your Memory?

- All forms of DRAM store data as charge in a capacitor
  - Charge will leak with time hence DRAM needs periodic refreshing to maintain data
  - Even so, may get occasional loss of charge due to ionisation or noise – modern chips even more so due to miniaturisation
  - If a single bit is changed from a "1" to a "0" then could change sign or order of magnitude of a number! Or it might be a fraction of a percent change in magnitude (harder to detect in code). Or it may change a target address for a jump, etc. All of which is bad news!

What Can be Done?

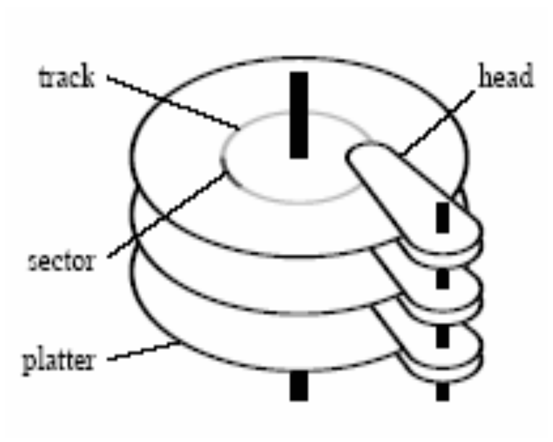
- Parity
  - Add an extra bit of memory to block of memory monitored which is "1" if an odd number of the checked bits are equal to "1" and "0" otherwise
  - Hence can detect a single bit corruption but cannot do anything except stop!
  - Typically 1 bit per byte => overhead 12.5%
- ECC (Error Correcting Code)
  - Adds more protection bits – typically one for every 2^n bits
  - Can detect and fix all single-bit errors, and detect all 2-bit errors and some 3-bit errors.
  - Cost is $2+\log_2(n)$, so 5 bits for 1-byte but 10 bits for 32 bytes (256 bits => overhead ~ 4%)
  - Essential as memory gets bigger and feature size shrinks!
  - See Google report on memory errors and real rates ...

Hard Disks / Solid State Drive

- Choice of just two internal interfaces these days
  - **PCIe** – Peripheral Component Interface Express – released in 2003 – serial, general purpose, 1 bit per lane, can have 1-32 bonded lanes, one device per end-point but can have switch to make multiple endpoints
  - **SATA** – serial, released in 2000, regularly updated (latest 2020). Only 1 device per channel. Also supports external devices too with eSATA since 2004.
  - Both come in different variants. PCIe generally faster.
- Also USB and FireWire for external devices
  - USB 3.0 spec finalised in Nov 2008 – first commercial external disks released in end-Sept 2009 . USB 4.0 spec in 2019 based on Thunderbolt 3.
  - Latest innovation is Thunderbolt (joint Intel + Apple)
  - MacBook Pro (Feb 2011) – designed to extend internal PCI Express bus to outside for video – but can also be used for disk
  - V2 released in 2013, V3 in 2015 – faster, bigger bandwidth, etc
- May find SCSI on old / high-end servers – serial & obsolete

Hard Disk Internals

- Single disk actually contains many platters, each with two magnetic surfaces (GMR is latest technology)
- Disk spins up to 15,000 RPM
- Extreme tolerances
  - head hovers 50 nm above surface (like flying a 747 airliner less than 0.2 cm above ground)
- Extreme materials engineering
  - Exponential growth in bit density – doubling every year at the moment
  - Work of Prof O'Grady etc



Best head seek time ~ 4 ms

Rotational latency ~ 2 ms

Bandwidth ~ 1000 MB/s

c.f. DRAM with 10 ns latency and 6 GB/s bandwidth

Hence need caching to keep up with memory

Solid State Drive

- Either NAND based (keeps data when switched off) or RAM (faster but volatile)
- Host interface can be SCSI, SATA, USB, etc.
- Advantages
  - no spin-up delay, low latency (less than 0.1 ms), high bandwidth (600 MB/s)
- Disadvantages
  - high cost (2x HD but coming down), smaller max capacity, higher risk of catastrophic failure, limited lifetime of each cell – need automatic "wear levelling"

Video

- Final subsystem of consequence is video
- Traditionally unimportant for HPC, except for scientific visualisation, but GPU is useful
  - Recent developments in GPU relieve CPU of much load – in some modern PCs the GPU has more transistors than the CPU! nVIDIA pioneered with CUDA …
- OpenGL is an open library for 3D graphics.
  - Pioneered by SGI with hardware support in its graphic workstations but early PC versions did it all in software and so were very slow.
  - Modern "3D-accelerators" do it in hardware and so fast.
  - Very useful for portable code! Adopted by Windows, Mac and UNIX, with many different language bindings.

Historical View (I)

- To understand current hardware, we need some knowledge of how current hardware evolved!
- Modern computing has its origins in the 1940s with early mainframes using valves as switches and ferrite cores for memory. The ideas developed have outlived the original hardware.
- By 1970, the concepts of disk drives, floating point, memory paging, parity protection, multitasking, caches, pipelining and out-of-order execution have all appeared in commercial systems, and high-level languages and wide-area-networking had been developed.

Historical View (II)

- In the 1970s the new ideas were vector computers, error-correcting memory and RISC. Unix, C and TeX were created, and Fortran was standardized.
- In the 1980s the first massively-parallel computers appeared, and the Internet is created.
- In the early 1990s the first multi-threaded CPUs were invented, MPI and OpenMP were standardized, and the WWW was created.
- Since then there has been little that is radically new – it has just been refinement of old ideas!

Further Reading

- "Computer Architecture – A Quantitative Approach (6th edition)", John L Hennessy and David A Patterson, (2017).
- https://en.wikipedia.org/wiki/Cache_placement_policies for cache policies etc
- Intel http://www.intel.com
- IBM http://www.ibm.com
- Seagate http://www.seagate.com
- Google memory study: http://www.cs.toronto.edu/~bianca/papers/sigmetrics09.pdf