CS 7172
Parallel and Distributed Computation

Overview

Kun Suo

Computer Science, Kennesaw State University

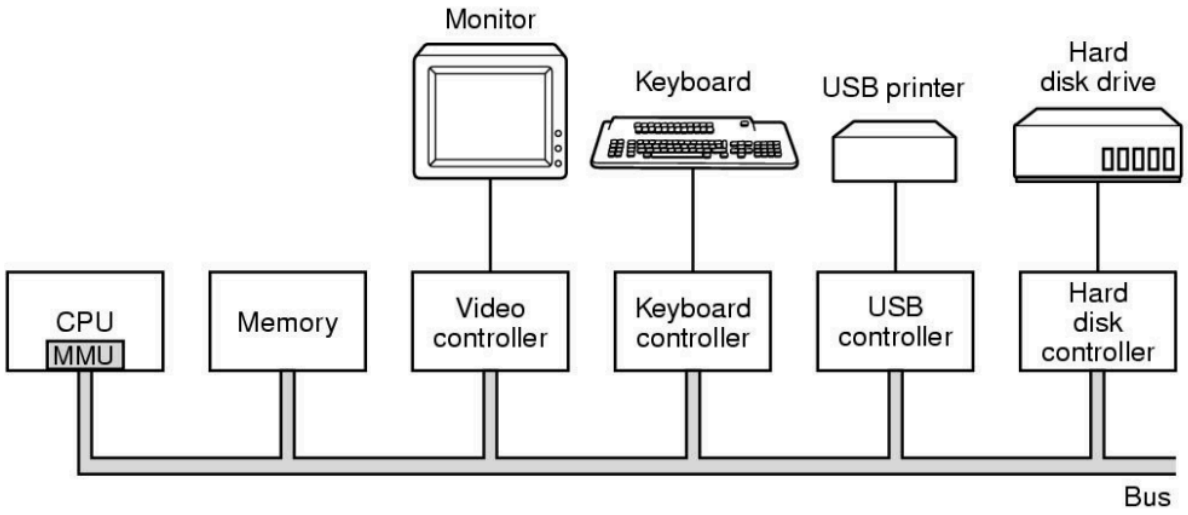
https://kevinsuo.github.io/

Outline

- Why we need ever-increasing performance.
- Why we're building parallel systems.
- Why we need to write parallel programs.
- How do we write parallel programs?
- What we'll be doing.
- Concurrent, parallel, distributed!

Computer Hardware Review

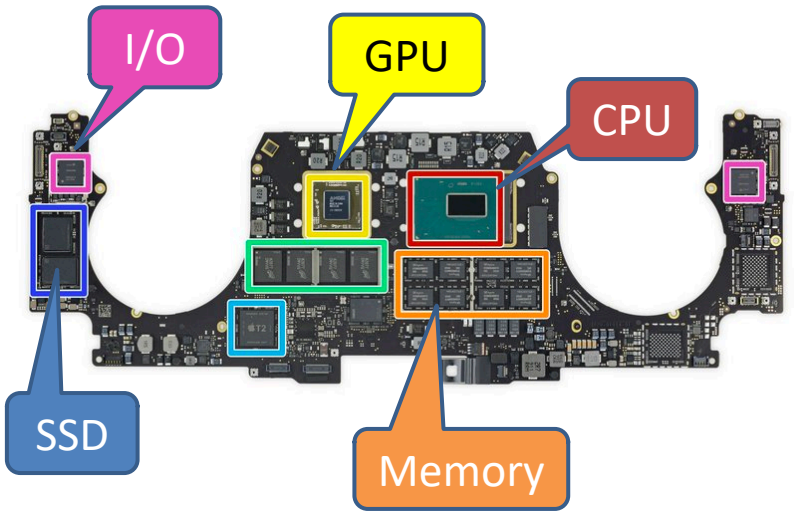
- Basic components of a simple personal computer



- CPU: data processing
- Memory: volatile data storage
- Disk: persistent data storage
- NIC: inter-machine communication
- Bus: intra-machine communication

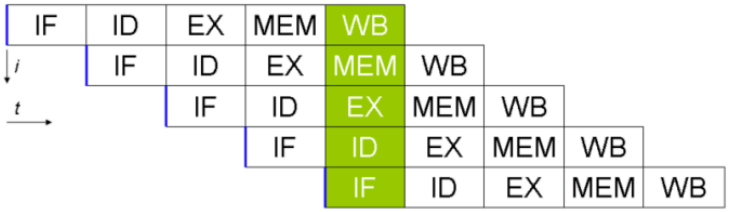
Computer Hardware Review

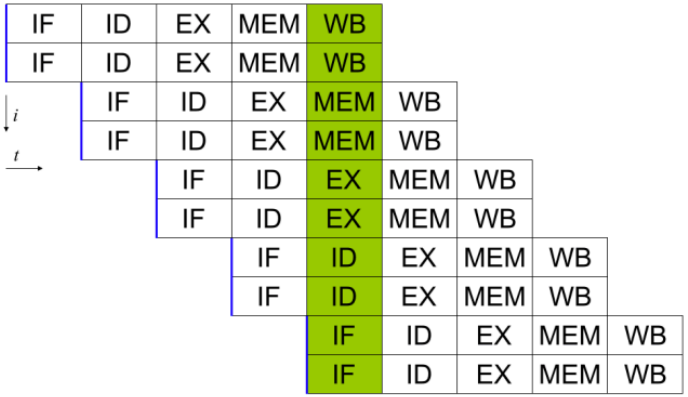
Computer Hardware Review

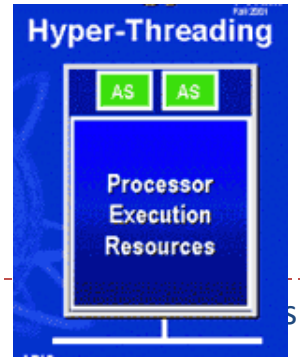


Central Processing Unit (CPU)

- Components
  - Arithmetic Logic Unit (ALU) -> Compute and data
  - Control Unit (CU) -> control device and system
- Clock rate
  - The speed at which a CPU is running
- Data storage
  - General-purpose registers: EAX, EBX …
  - Special-purpose registers: PC (program counter), SP (stack), IR (instruction register) …
- Parallelism
  - Instruction-level parallelism
  - Thread-level parallelism
    - Hyper-threading: duplicate units that store architectural states
    - Replicated: registers. Partitioned: ROB, load buffer… Shared: reservation station, caches

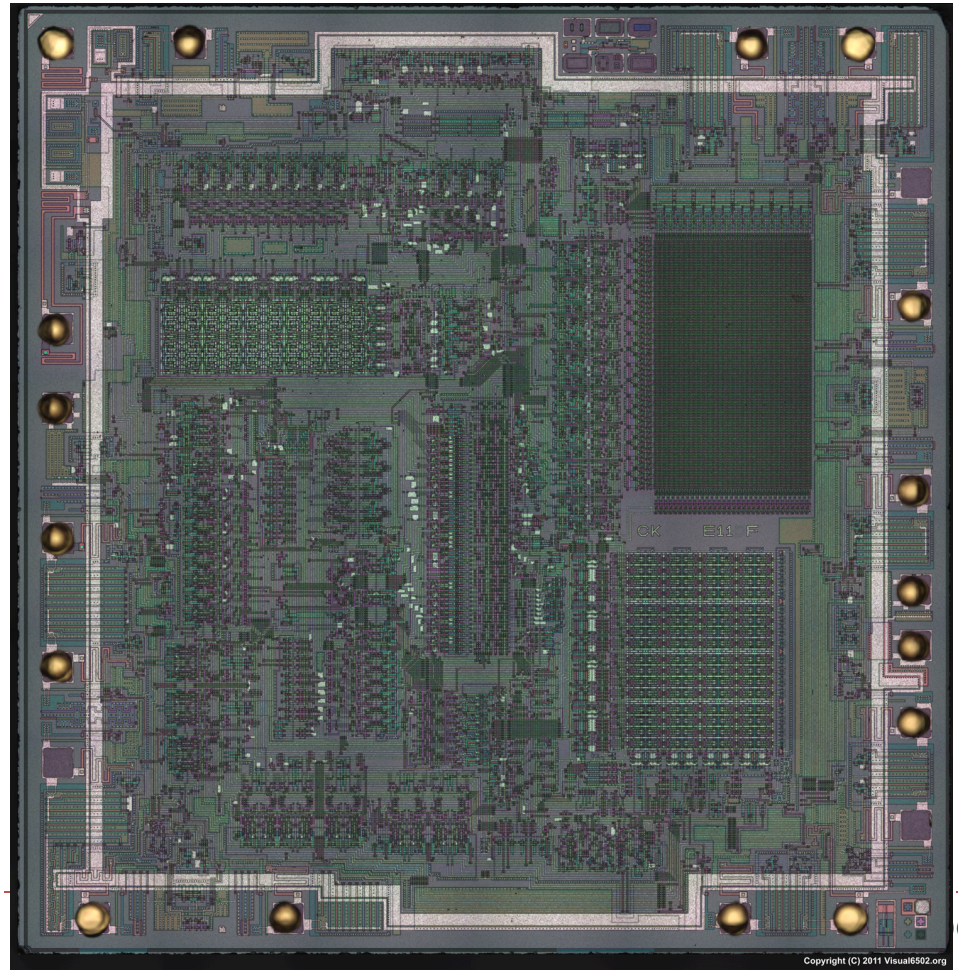






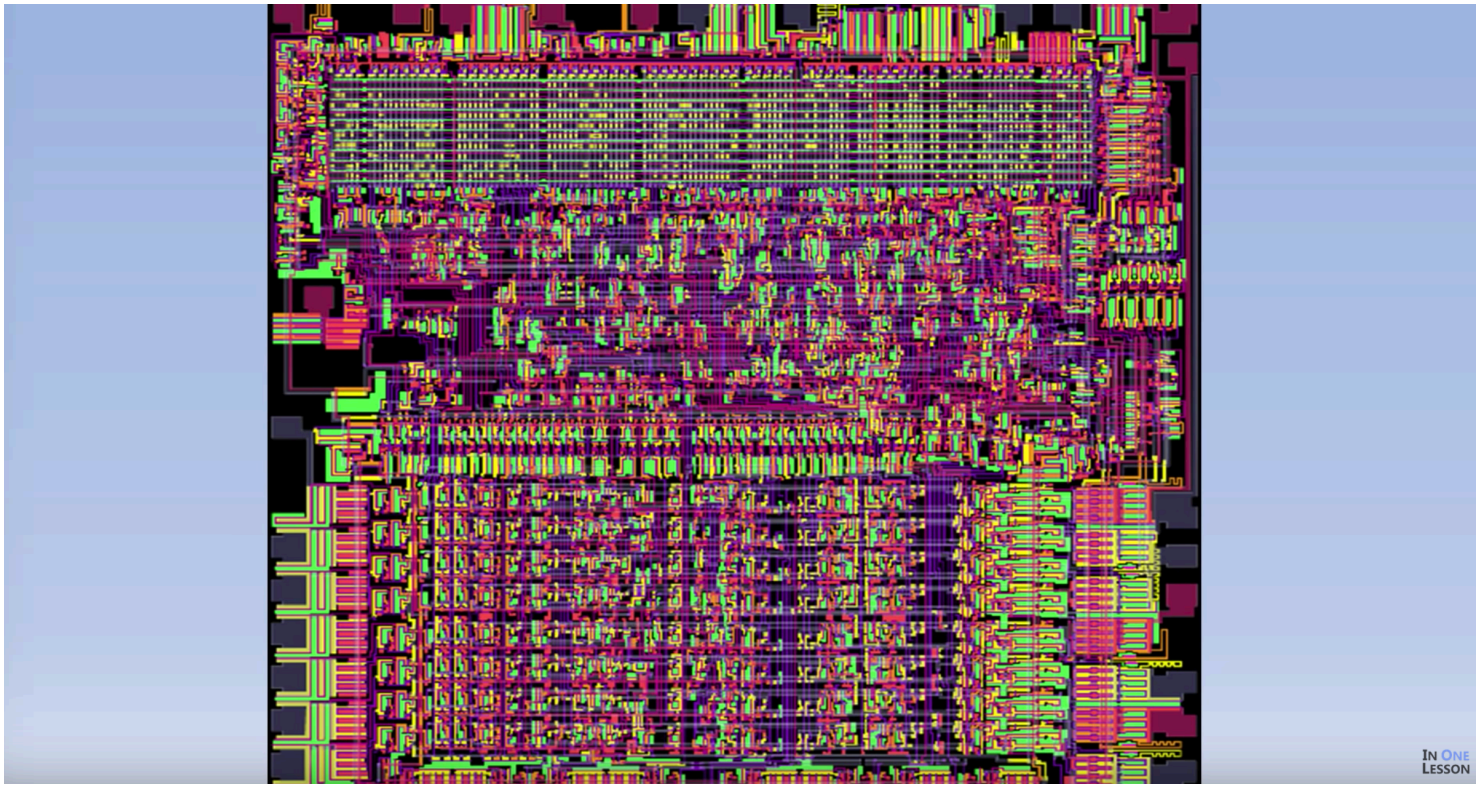
What's inside of CPU?

- http://www.visual6502.org/

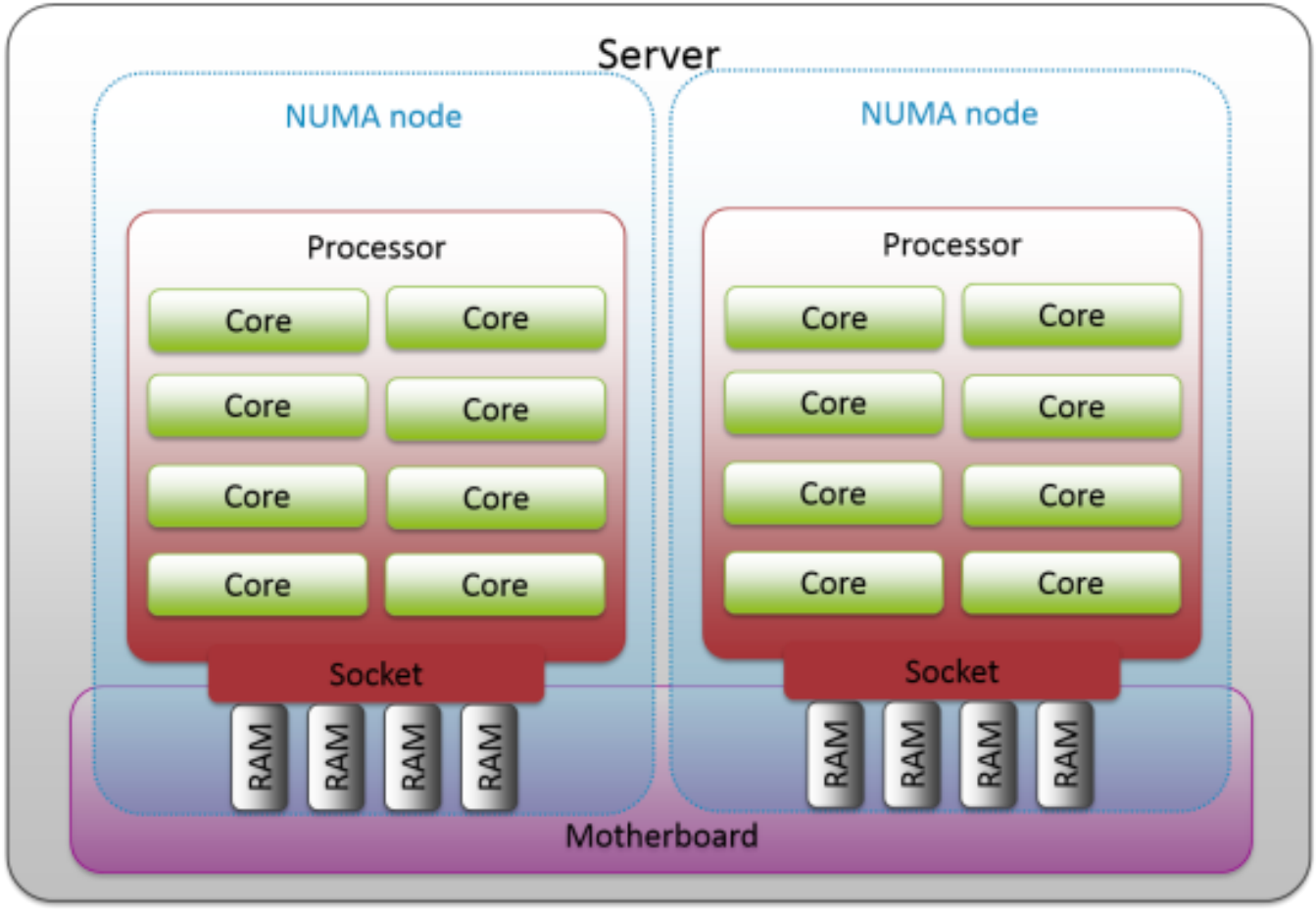


How CPU works?

- https://youtu.be/cNN_tTXABUA?t=494



NUMA node vs Socket vs Core relationship



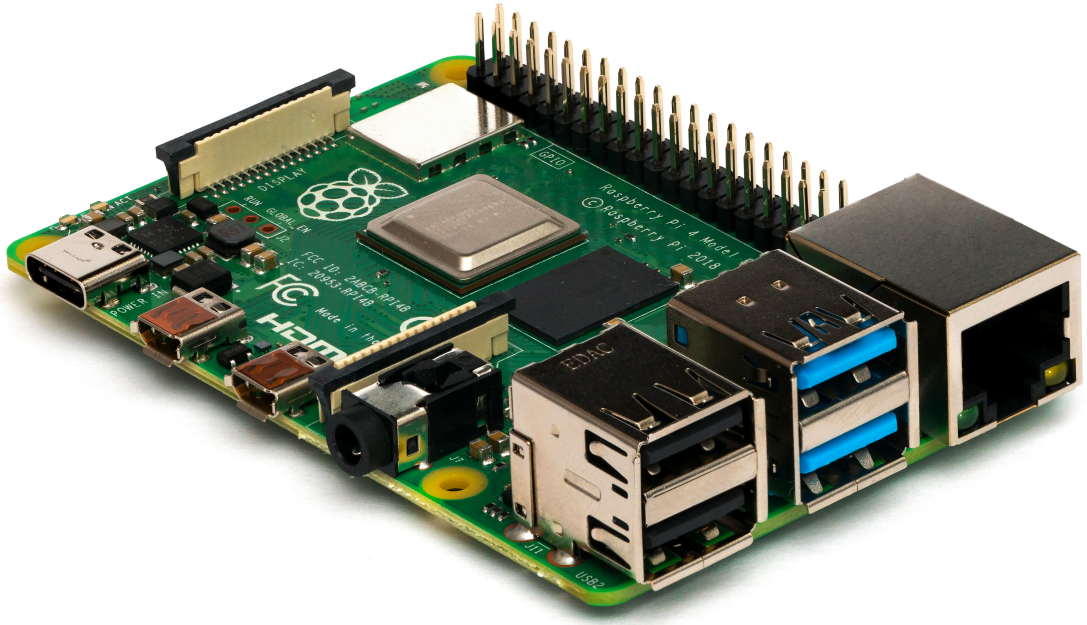
CPU information

- lscpu


```
administrator@ubuntuvm-1604 ~> lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                2
On-line CPU(s) list:   0,1
Thread(s) per core:    1
Core(s) per socket:    1
Socket(s):             2
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 79
Model name:            Intel(R) Xeon(R) CPU E5-2698 v4 @ 2.20GHz
Stepping:              1
CPU MHz:               2199.998
BogoMIPS:              4399.99
Hypervisor vendor:     VMware
Virtualization type:   full
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              51200K
NUMA node0 CPU(s):     0,1
Flags:                 fpu vme de pse tsc msr pae mce cx8 apic sep m
all nx pdpe1gb rdtscp lm constant_tsc arch_perfmon nopl xtopology ts
id sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave av
lt invpcid_single pti ssbd ibrs ibpb stibp fsgsbase tsc_adjust bmi1
 flush_l1d arch_capabilities
```

CPU information

- lscpu



```
pi@raspberrypi:~ $ lscpu
Architecture:            armv7l
Byte Order:              Little Endian
CPU(s):                  4
On-line CPU(s) list:     0-3
Thread(s) per core:      1
Core(s) per socket:      4
Socket(s):               1
Vendor ID:               ARM
Model:                   3
Model name:              Cortex-A72
Stepping:                r0p3
CPU max MHz:             1500.0000
CPU min MHz:             600.0000
BogoMIPS:                108.00
Flags:                   half thumb fastmult vfp edsp neon vfpv3 tls vfpv4 idiva idivt vfpd32 lpae evtstrm crc32
```

CPU information

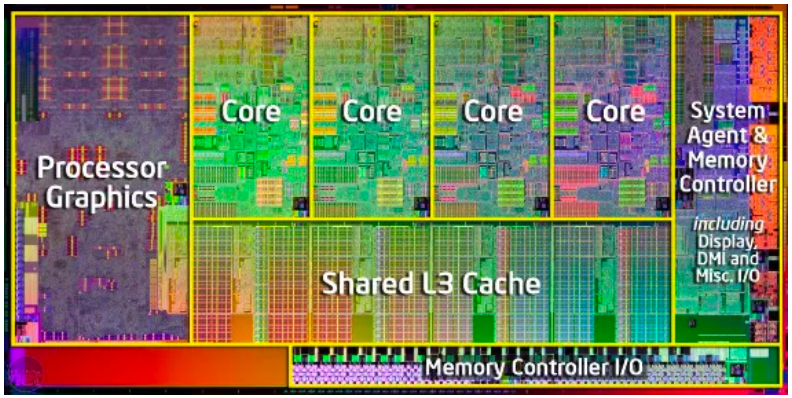
- $ cat /proc/cpuinfo

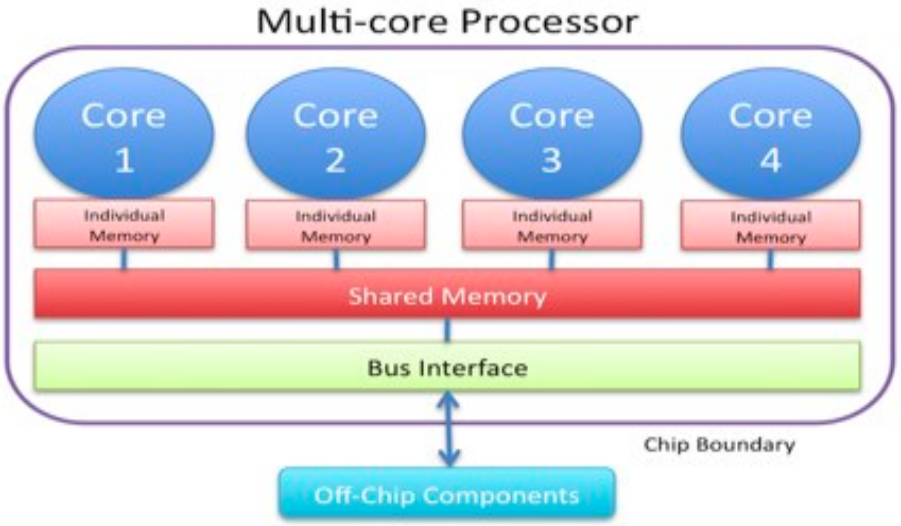
```
administrator@ubuntuvm-1604 ~> cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 79
model name      : Intel(R) Xeon(R) CPU E5-2698 v4 @ 2.20GHz
stepping        : 1
microcode       : 0xb000036
cpu MHz         : 2199.998
cache size      : 51200 KB
physical id     : 0
siblings        : 1
core id         : 0
cpu cores       : 1
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 20
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep
x pdpe1gb rdtscp lm constant_tsc arch_perfmon nopl xtopology t
e4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave a
vpcid_single pti ssbd ibrs ibpb stibp fsgsbase tsc_adjust bmi1
h_l1d arch_capabilities
bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_stor
bogomips        : 4399.99
clflush size    : 64
cache_alignment : 64
address sizes   : 42 bits physical, 48 bits virtual
power management:

processor       : 1
vendor_id       : GenuineIntel
cpu family      : 6
model           : 79
model name      : Intel(R) Xeon(R) CPU E5-2698 v4 @ 2.20GHz
stepping        : 1
microcode       : 0xb000036
cpu MHz         : 2199.998
cache size      : 51200 KB
physical id     : 2
siblings        : 1
core id         : 0
cpu cores       : 1
apicid          : 2
initial apicid  : 2
fpu             : yes
fpu_exception   : yes
cpuid level     : 20
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep
x pdpe1gb rdtscp lm constant_tsc arch_perfmon nopl xtopology
e4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave
vpcid_single pti ssbd ibrs ibpb stibp fsgsbase tsc_adjust bmi
h_l1d arch_capabilities
bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_st
bogomips        : 4399.99
clflush size    : 64
cache_alignment : 64
address sizes   : 42 bits physical, 48 bits virtual
power management:
```

Multi-Core Processors (SMP)

- Multiple CPUs on a single chip

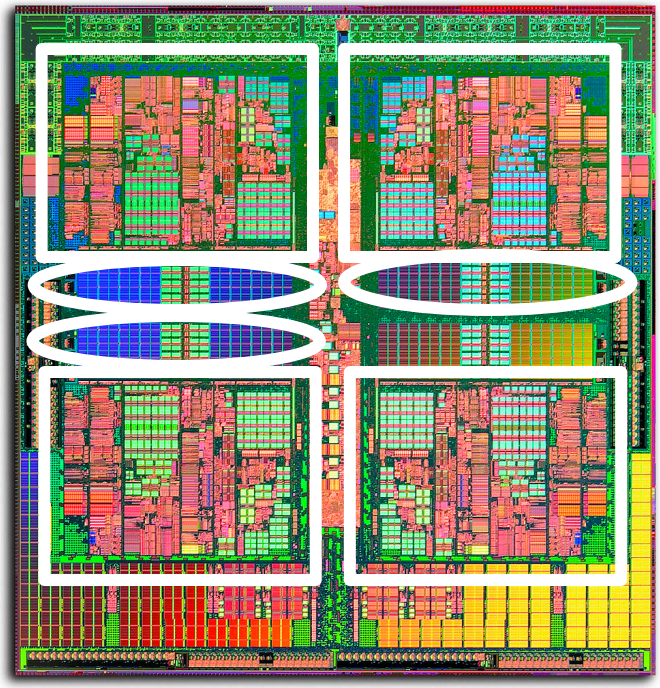


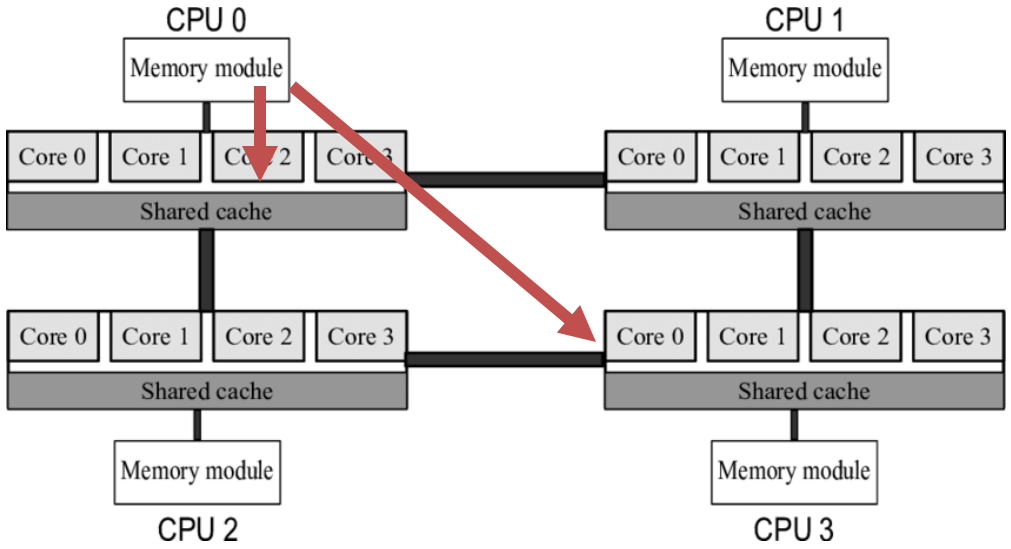


Symmetric multiprocessing (**SMP**)

Multi-Core Processors (NUMA)

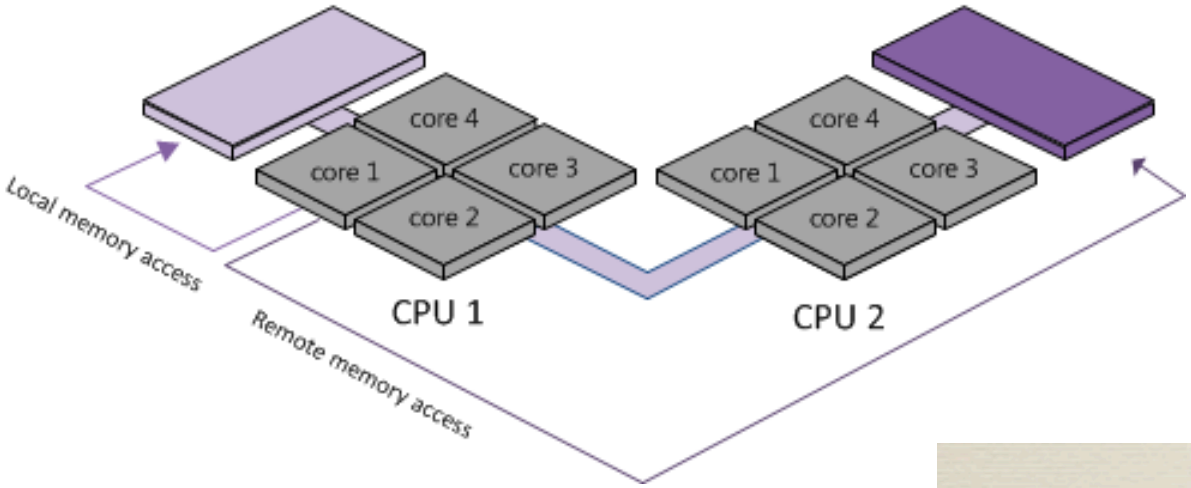
- Multiple CPUs on a single chip

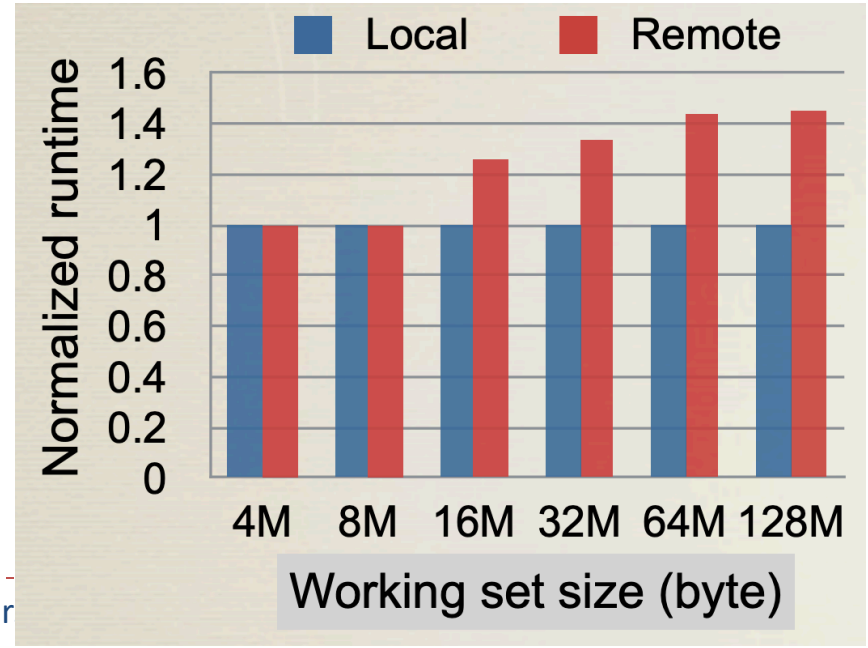




Non-uniform memory access (**NUMA**)

Multi-Core Processors (NUMA)





Check CPU topology

- $ likwid-topology –g

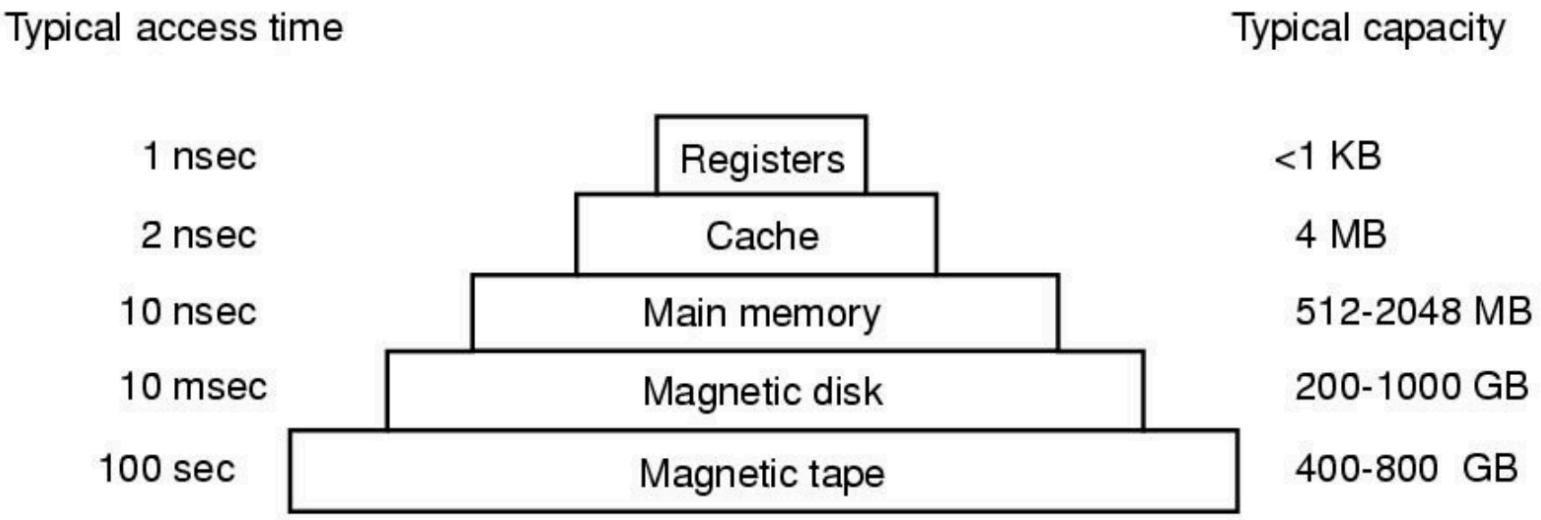
```
ksuo@ksuo-VirtualBox ~/likwid-5.0.0> likwid-topology -g
--------------------------------------------------------------------------------
CPU name:       Intel(R) Core(TM) i9-9880H CPU @ 2.30GHz
CPU type:       Intel Coffeelake processor
CPU stepping:   13
********************************************************************************
Hardware Thread Topology
********************************************************************************
Sockets:                1
Cores per socket:       4
Threads per core:       1
--------------------------------------------------------------------------------
HWThread        Thread          Core            Socket          Available
0               0               0               0               *
1               0               1               0               *
2               0               2               0               *
3               0               3               0               *
--------------------------------------------------------------------------------
Socket 0:               ( 0 1 2 3 )
--------------------------------------------------------------------------------
********************************************************************************
Graphical Topology
********************************************************************************
Socket 0:
+-------------------------------------------+
| +--------+ +--------+ +--------+ +--------+ |
| |    0   | |    1   | |    2   | |    3   | |
| +--------+ +--------+ +--------+ +--------+ |
| +--------+ +--------+ +--------+ +--------+ |
| |  32 kB | |  32 kB | |  32 kB | |  32 kB | |
| +--------+ +--------+ +--------+ +--------+ |
| +--------+ +--------+ +--------+ +--------+ |
| | 256 kB | | 256 kB | | 256 kB | | 256 kB | |
| +--------+ +--------+ +--------+ +--------+ |
| +--------+ +--------+ +--------+ +--------+ |
| |  16 MB | |  16 MB | |  16 MB | |  16 MB | |
| +--------+ +--------+ +--------+ +--------+ |
+-------------------------------------------+
```

NUMA example:

https://github.com/RRZE-HPC/likwid/wiki/TutorialNUMA

Memory

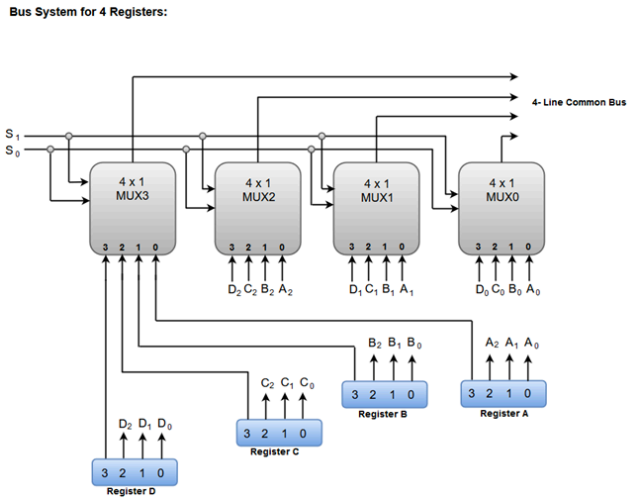
- A typical memory hierarchy

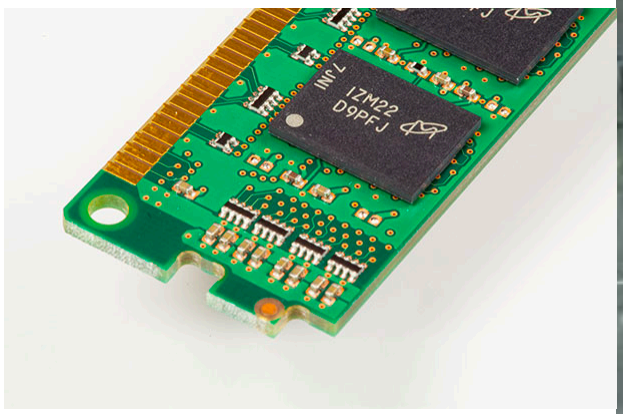


Minimize the access time vs. Cost

Memory

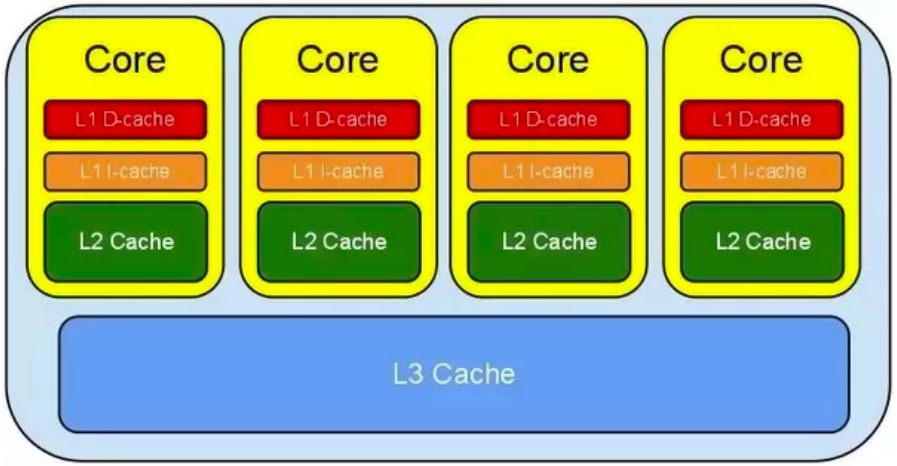
- A typical memory hierarchy

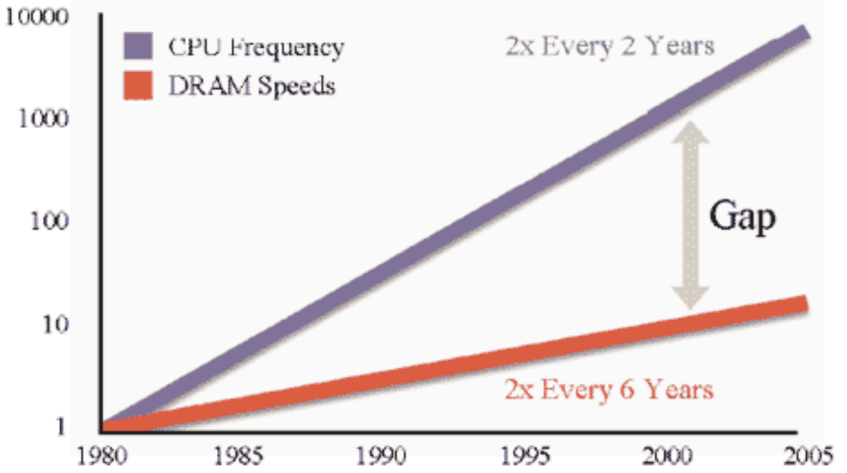




Cache

- Why Cache is important?





A larger size than registers
A much faster speed than memory
Tradeoff between performance and cost

MacBook Pro

Hardware Overview:

Model Name:	MacBook Pro
Model Identifier:	MacBookPro15,1
Processor Name:	Intel Core i9
Processor Speed:	2.3 GHz
Number of Processors:	1
Total Number of Cores:	8
L2 Cache (per Core):	256 KB
L3 Cache:	16 MB
Hyper-Threading Technology:	Enabled
Memory:	16 GB
Boot ROM Version:	220.270.99.0.0 (iBridge: 16.16.6568.0.0,0)
Serial Number (system):	C02YR4JHLVCJ
Hardware UUID:	DCC2D30A-9630-57B5-89A2-5F2B85254DC1

Check Cache info

- $ likwid-topology –g
- $ lscpu | grep cache

```
********************************************************************************
Cache Topology
********************************************************************************
Level:                  1
Size:                   32 kB
Cache groups:           ( 0 ) ( 1 ) ( 2 ) ( 3 )
--------------------------------------------------------------------------------
Level:                  2
Size:                   256 kB
Cache groups:           ( 0 ) ( 1 ) ( 2 ) ( 3 )
--------------------------------------------------------------------------------
Level:                  3
Size:                   16 MB
Cache groups:           ( 0 ) ( 1 ) ( 2 ) ( 3 )
--------------------------------------------------------------------------------
********************************************************************************
NUMA Topology
********************************************************************************
NUMA domains:           1
--------------------------------------------------------------------------------
Domain:                 0
Processors:             ( 0 1 2 3 )
Distances:              10
Free memory:            1967.79 MB
Total memory:           3942.19 MB
--------------------------------------------------------------------------------
```

```
ksuo@ksuo-VirtualBox ~/likwid-5.0.0> lscpu | grep cache
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              16384K
```

Memory information

- free

```
administrator@ubuntuvm-1604 ~> free
              total        used        free      shared  buff/cache   available
Mem:        8168756     1348160     3031888       97424     3788708     6383036
Swap:        998396           0      998396
```

- The free command displays:
  - ✓ Total amount of free and used physical memory
  - ✓ Total amount of swap memory in the system
  - ✓ Buffers and caches used by the kernel

Memory information

- cat /proc/meminfo
  - MemTotal
  - MemFree
  - MemAvailable
  - Buffers
  - Cached
  - SwapCached
  - SwapTotal
  - SwapFree

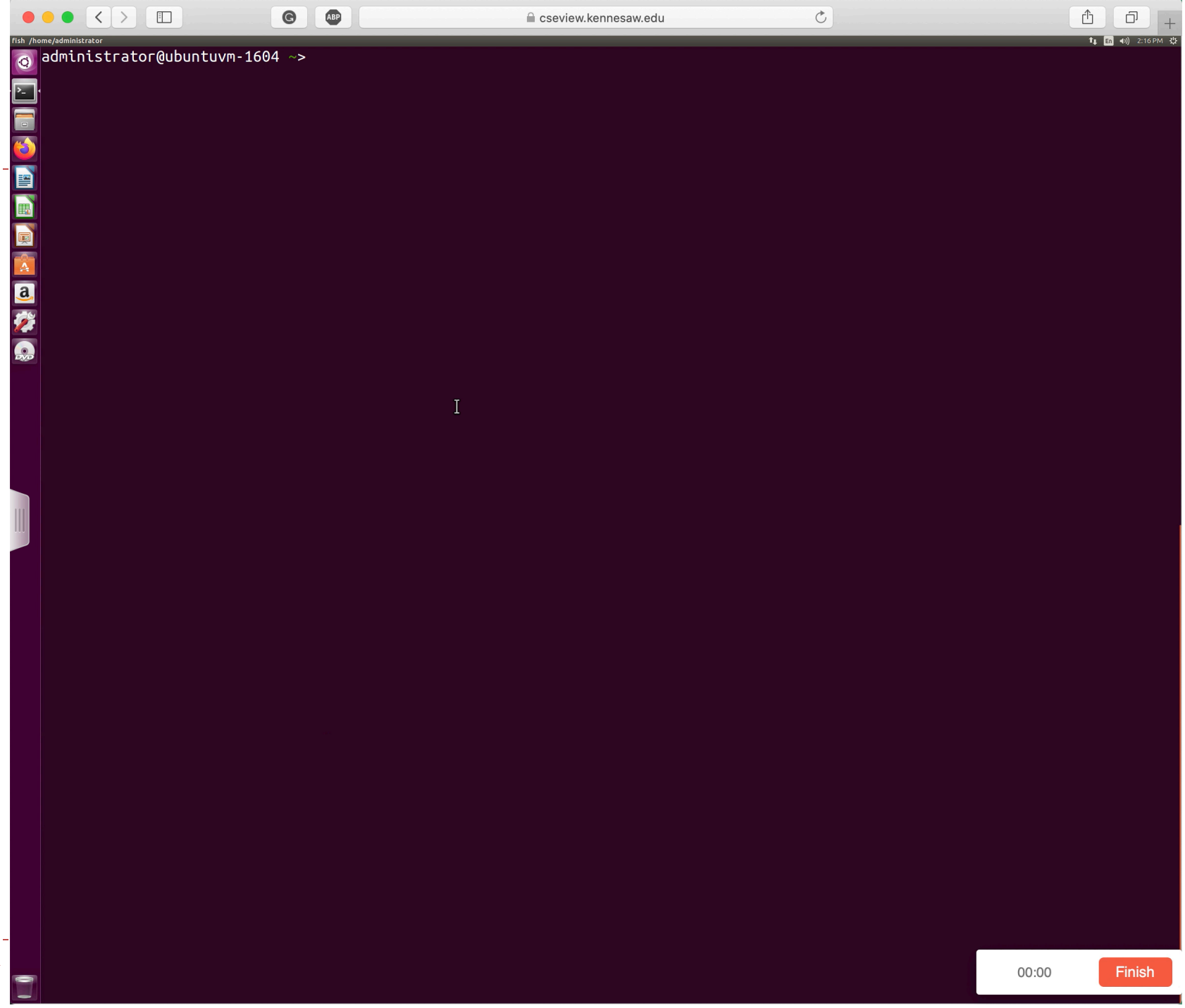
```
administrator@ubuntuvm-1604 ~> cat /proc/meminfo
MemTotal:        8168756 kB
MemFree:         3034596 kB
MemAvailable:    6385744 kB
Buffers:          287636 kB
Cached:          3188812 kB
SwapCached:            0 kB
Active:          2523764 kB
Inactive:        2177968 kB
Active(anon):    1226312 kB
Inactive(anon):    96392 kB
Active(file):    1297452 kB
Inactive(file):  2081576 kB
Unevictable:          48 kB
Mlocked:              48 kB
SwapTotal:       998396 kB
SwapFree:        998396 kB
Dirty:              264 kB
Writeback:            0 kB
AnonPages:       1225344 kB
Mapped:           304212 kB
Shmem:             97424 kB
Slab:             312268 kB
SReclaimable:     278104 kB
SUnreclaim:        34164 kB
KernelStack:        8896 kB
PageTables:        31004 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:     5082772 kB
Committed_AS:    5183160 kB
VmallocTotal:   34359738367 kB
VmallocUsed:           0 kB
VmallocChunk:          0 kB
HardwareCorrupted:     0 kB
AnonHugePages:         0 kB
ShmemHugePages:        0 kB
ShmemPmdMapped:        0 kB
CmaTotal:              0 kB
CmaFree:               0 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
DirectMap4k:      135040 kB
DirectMap2M:     5107712 kB
DirectMap1G:     5242880 kB
```

Memory information

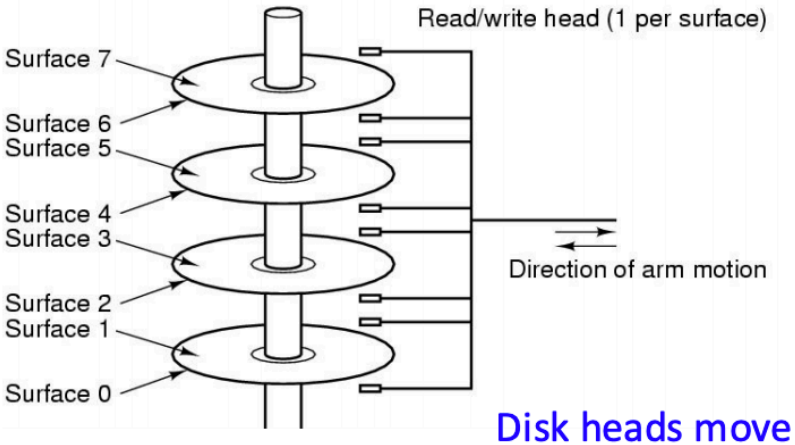
- lstopo: show the CPU cache and logical CPU layout

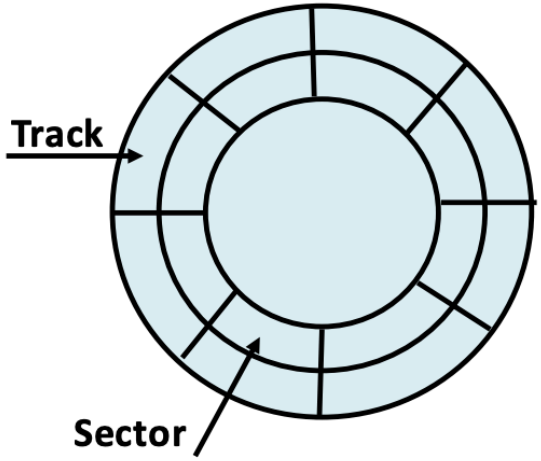
- Intall: $ sudo apt-get install hwloc

- Run: $ lstopo


cseview.kennesaw.edu
fish /home/administrator
2:16 PM
administrator@ubuntuvm-1604 ~>
00:00
Finish

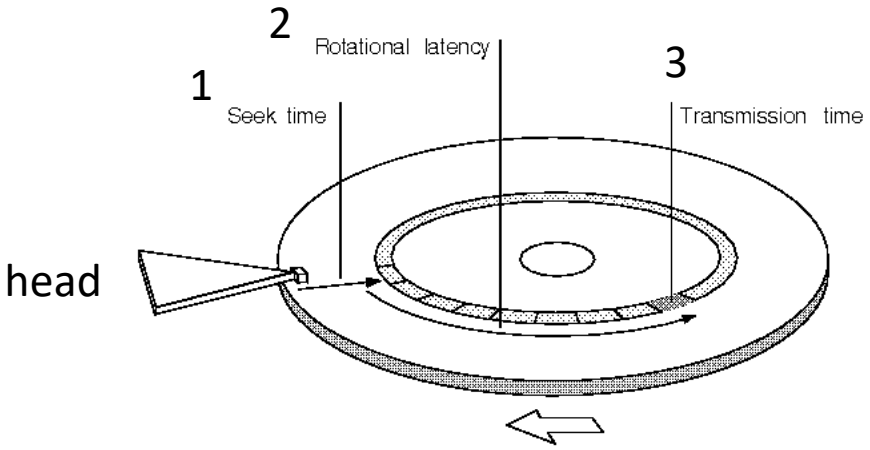
Disk





- A stack of platters, a surface with a magnetic coating
- Typical numbers (depending on the disk size):
  - 500 to 2,000 tracks per surface
  - 32 to 128 sectors per track
    - A sector is the smallest unit that can be read or written
- Originally, all tracks have the same number of sectors

Disk



- Disk head: each side of a platter has separate disk head
- Read/write data is a three-stage process:
  - Seek time: position the arm over the proper track
  - Rotational latency: wait for the desired sector to rotate under the read/write head
  - Transfer time: transfer a block of bits (sector) under the read-write head
- Average seek time as reported by the industry:
  - Typically in the range of 8 ms to 15 ms

Disk information

- lsblk
  - Lists out all the storage blocks, which includes disk partitions and optical drives. Details include the total size of the partition/block and the mount point if any.

```
administrator@ubuntuvm-1604 ~> lsblk
NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sr0     11:0    1  1.5G  0 rom  /media/administrator/Ubuntu 16.04.5 LTS amd64
sda      8:0    0   40G  0 disk
├─sda2   8:2    0    1K  0 part
├─sda5   8:5    0  975M  0 part [SWAP]
└─sda1   8:1    0   39G  0 part /
```

Disk information

- df
  - prints out details about only mounted file systems. The list generated by df even includes file systems that are not real disk partitions.

```
administrator@ubuntuvm-1604 ~> df
Filesystem     1K-blocks    Used Available Use% Mounted on
udev             4054632       0   4054632   0% /dev
tmpfs             816876   42148    774728   6% /run
/dev/sda1       40168028 7350596  30753972  20% /
tmpfs            4084376    1044   4083332   1% /dev/shm
tmpfs               5120       0      5120   0% /run/lock
tmpfs            4084376       0   4084376   0% /sys/fs/cgroup
/dev/sr0         1610928 1610928         0 100% /media/administrator/Ubuntu 16.04.5 LTS amd64
tmpfs             816876      56    816820   1% /run/user/1000
```

Disk information

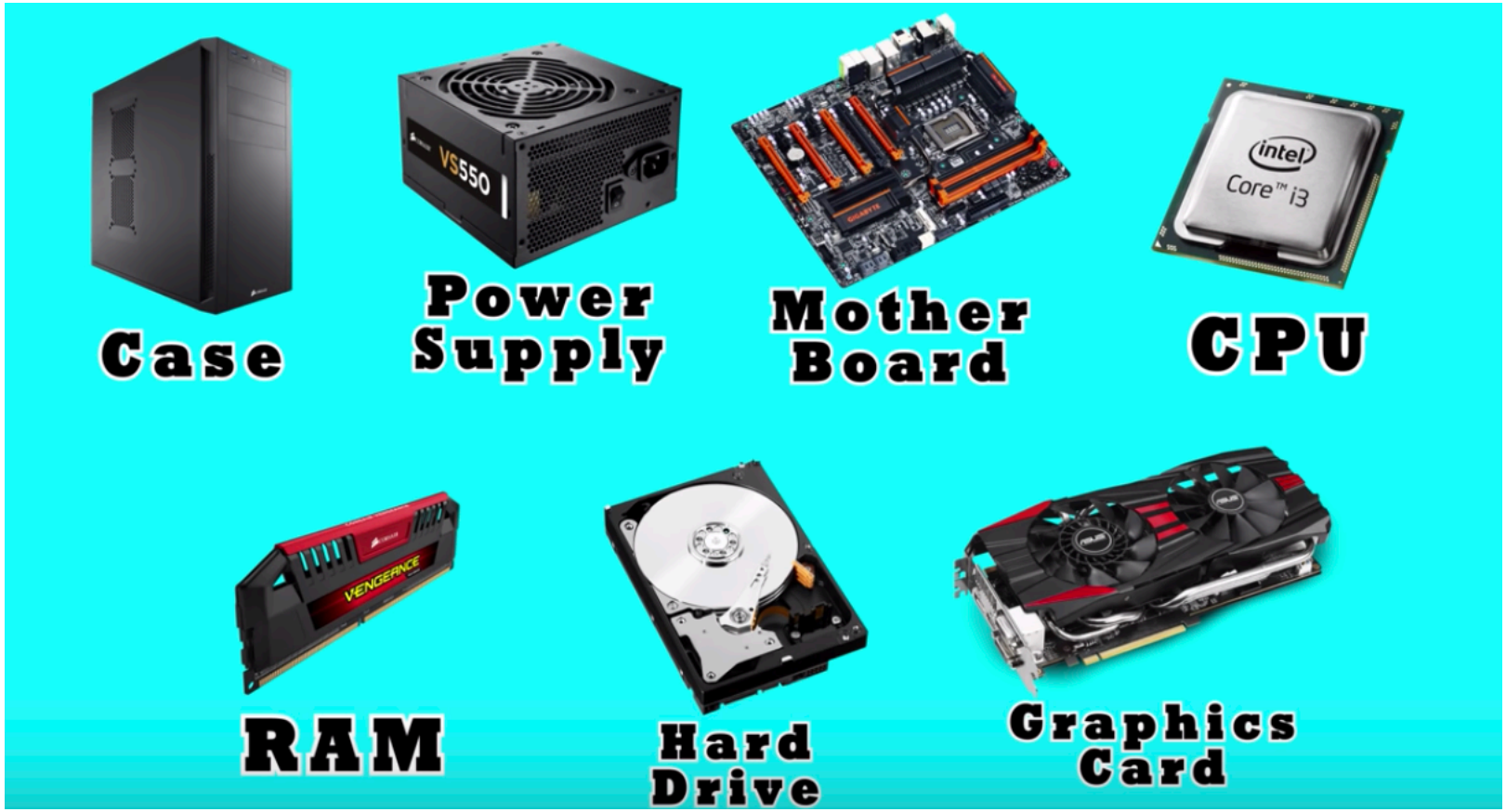
- fdisk
  - display the partitions and details like file system type

```
administrator@ubuntuvm-1604 ~> sudo fdisk -l
[sudo] password for administrator:
Disk /dev/sda: 40 GiB, 42949672960 bytes, 83886080 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x6e6dc012

Device     Boot    Start      End  Sectors  Size Id Type
/dev/sda1  *        2048 81885183 81883136   39G 83 Linux
/dev/sda2       81887230 83884031  1996802  975M  5 Extended
/dev/sda5       81887232 83884031  1996800  975M 82 Linux swap / Solaris
```

Disk R/W Process

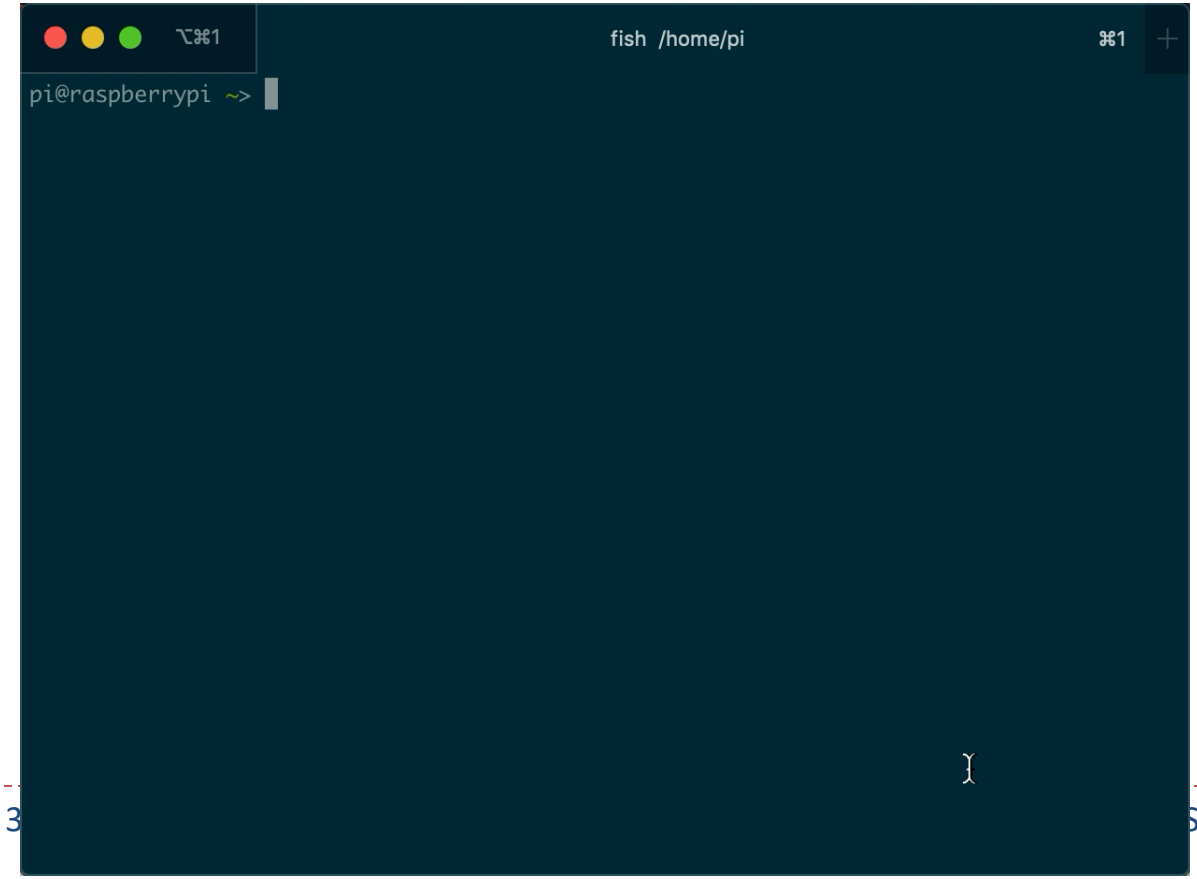
An interesting video introducing hardware



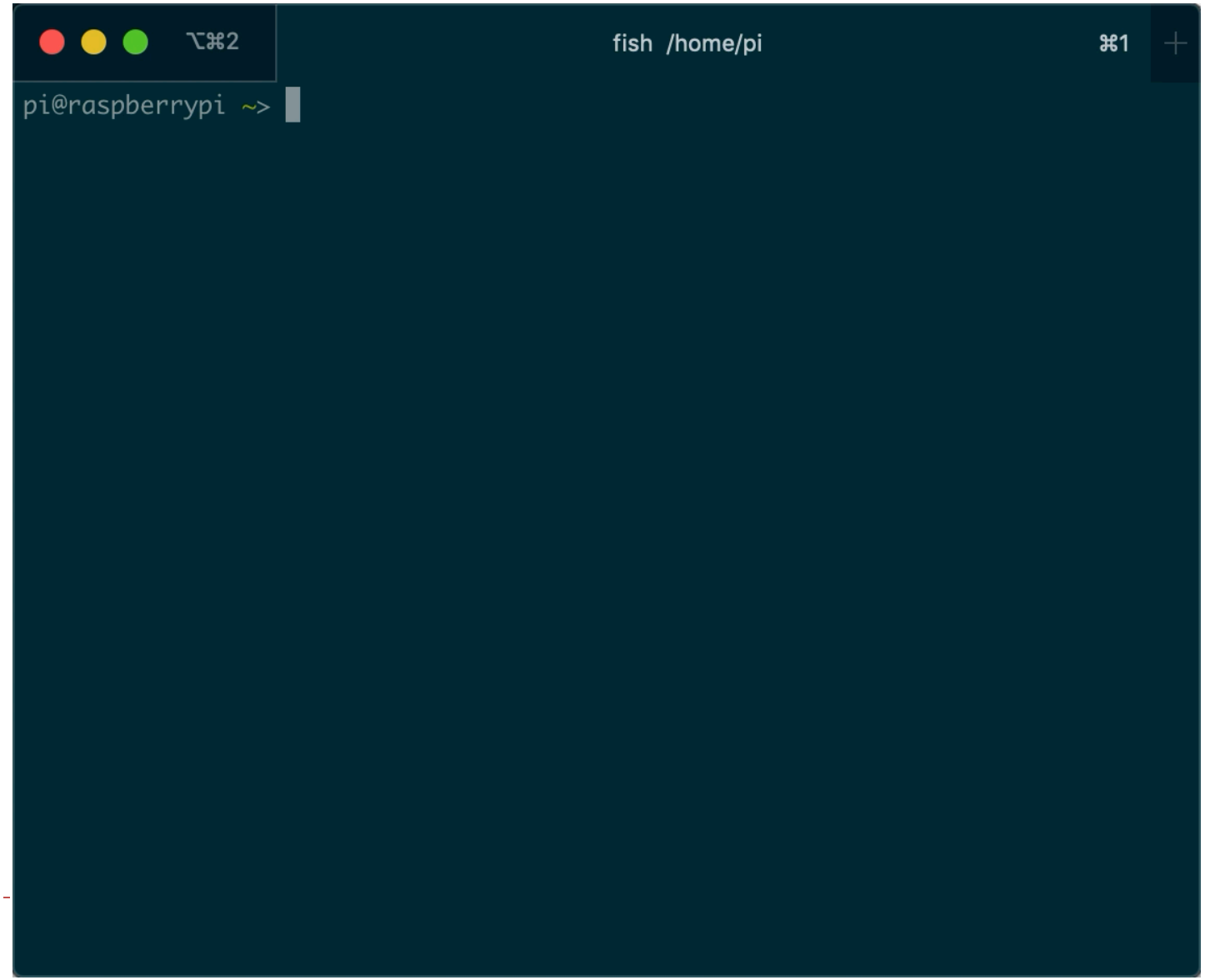
https://www.youtube.com/watch?v=ExxFxD4OSZ0

How to get my machine spec?

- Inxi: https://www.tecmint.com/inxi-command-to-find-linux-system-information/



How to get my machine spec? One command for All!

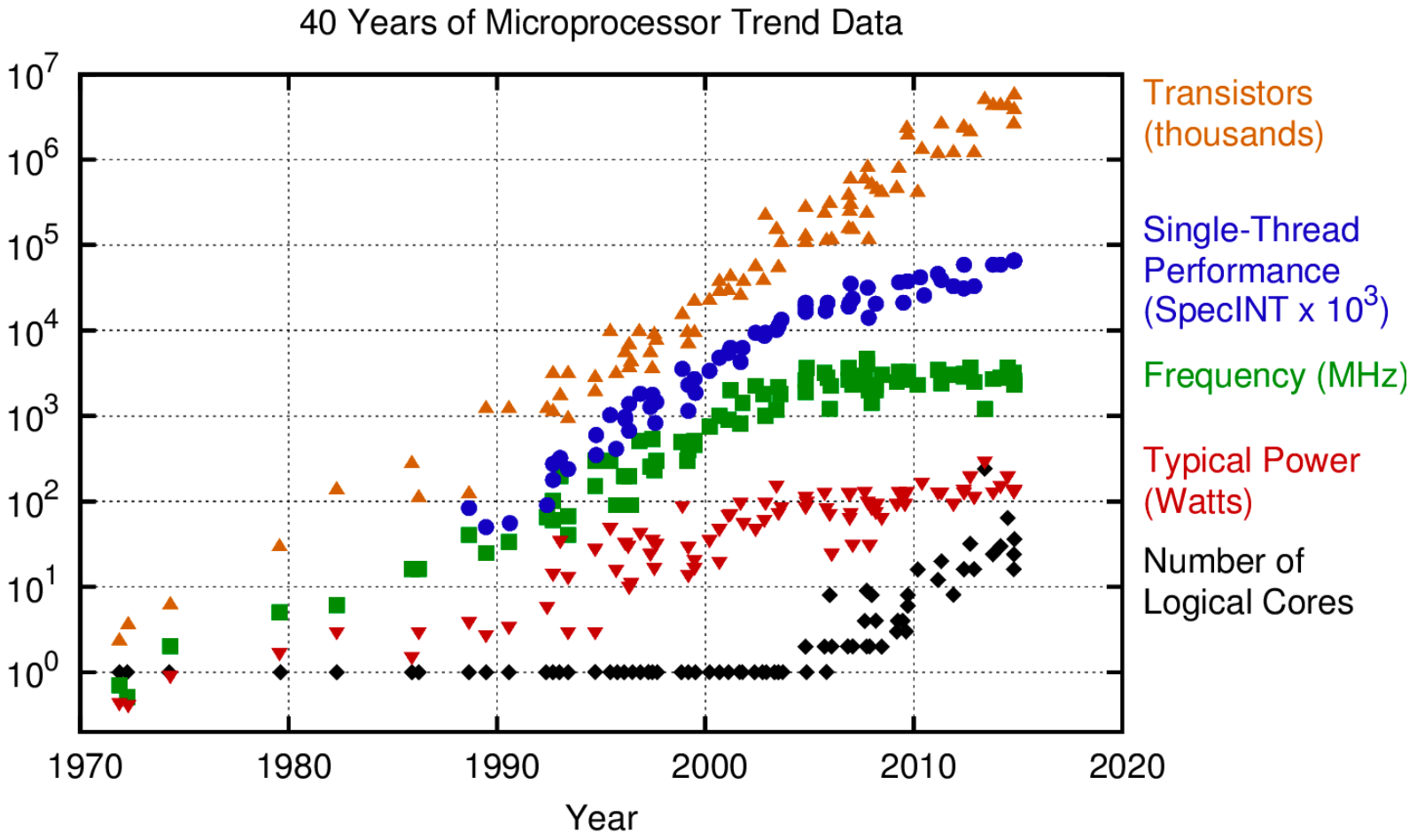


Changing times

- From 1986 – 2002, microprocessors were speeding like a rocket, increasing in performance an average of 50% per year.

- Since then, it's dropped to about 20% increase per year.

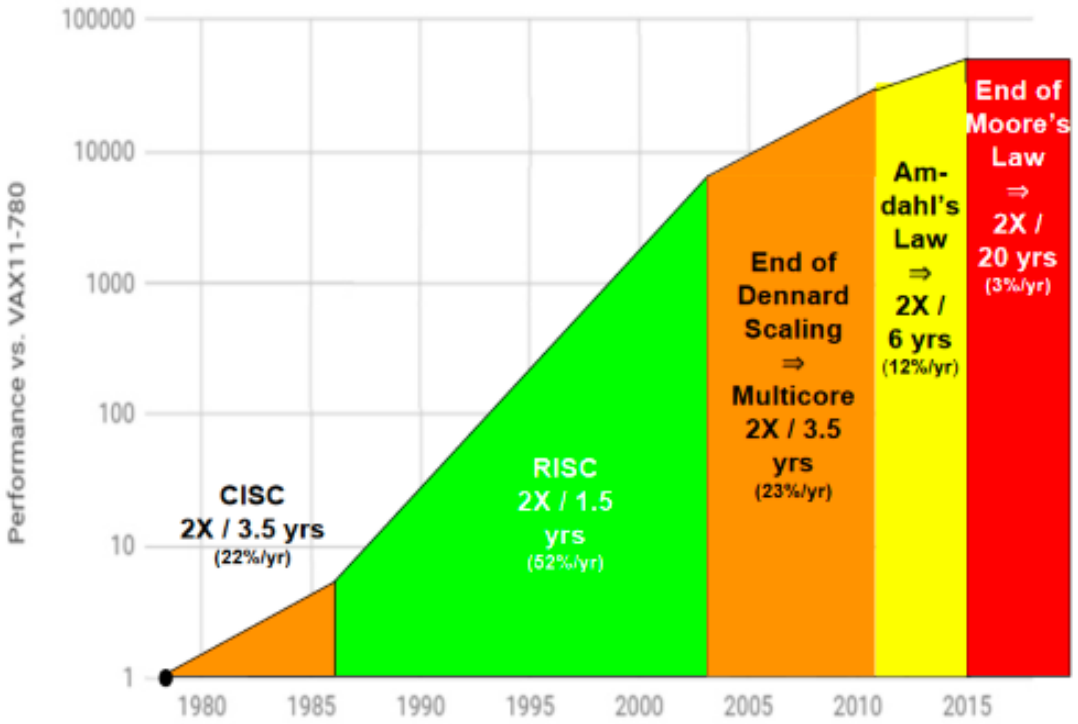
Changing times



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2015 by K. Rupp

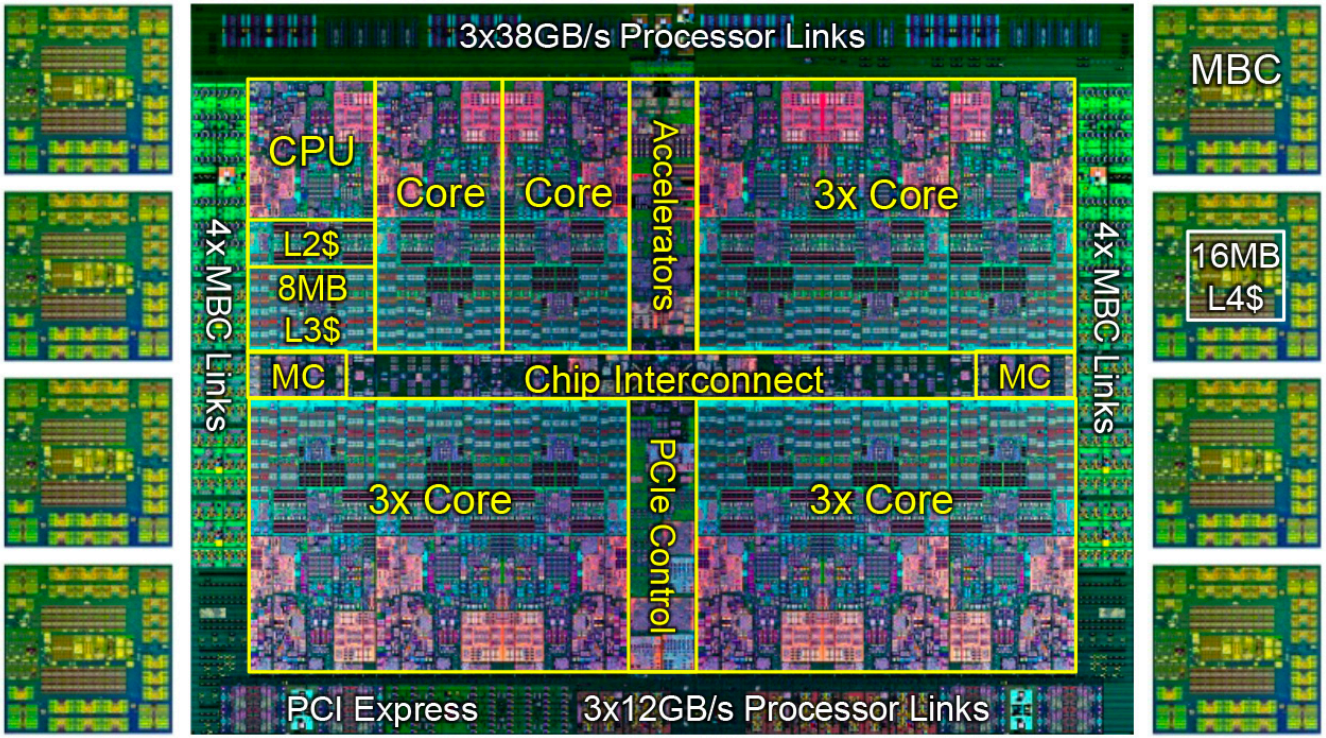
Moore's Law

- The observation that the number of transistors in a dense integrated circuit doubles about every two years



An intelligent solution

- Instead of designing and building faster microprocessors, put multiple processors on a single integrated circuit.



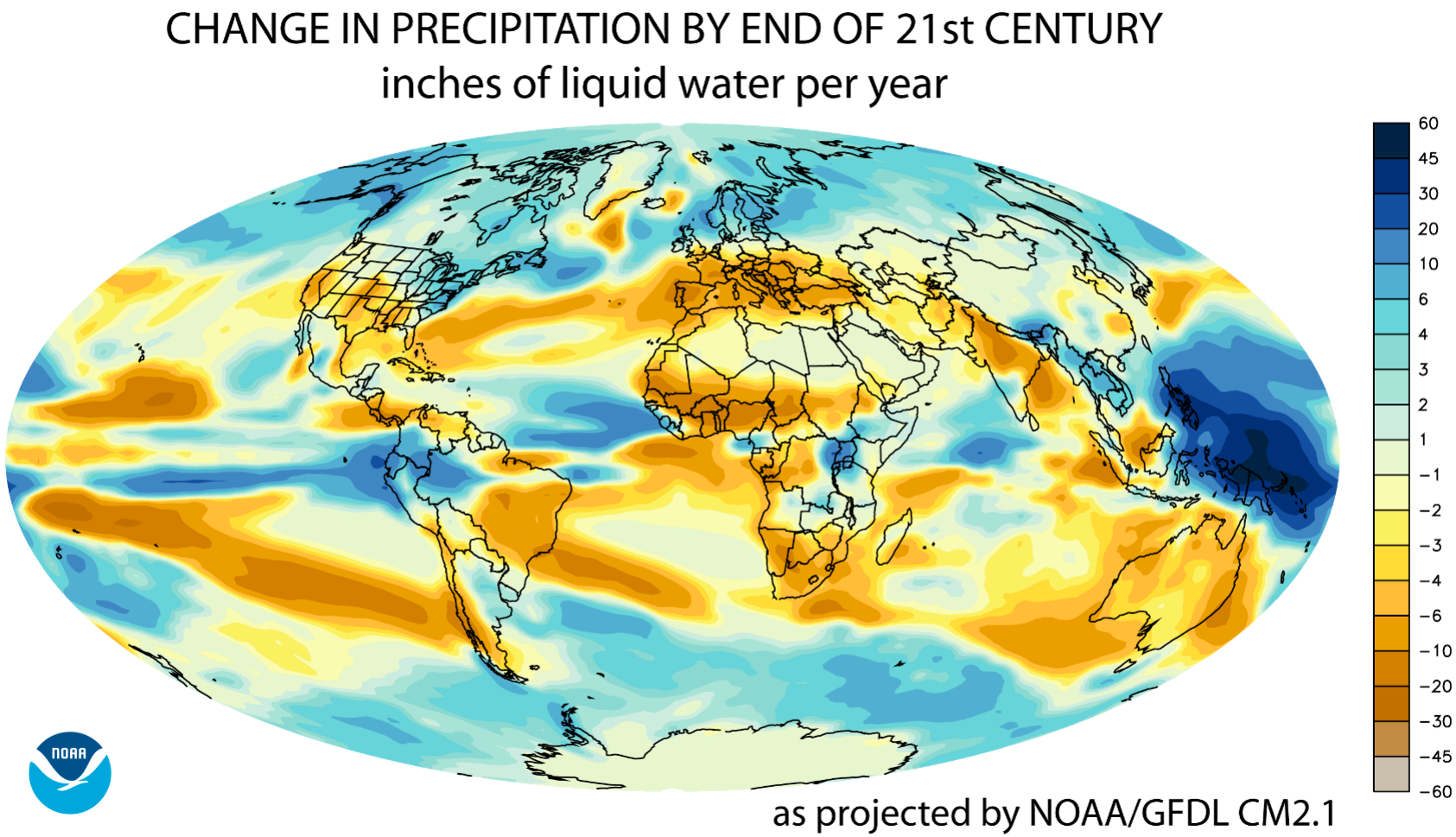
Now it's up to the programmers

- Adding more processors doesn't help much if programmers aren't aware of them…
- … or don't know how to use them.
- Serial programs don't benefit from this approach (in most cases).

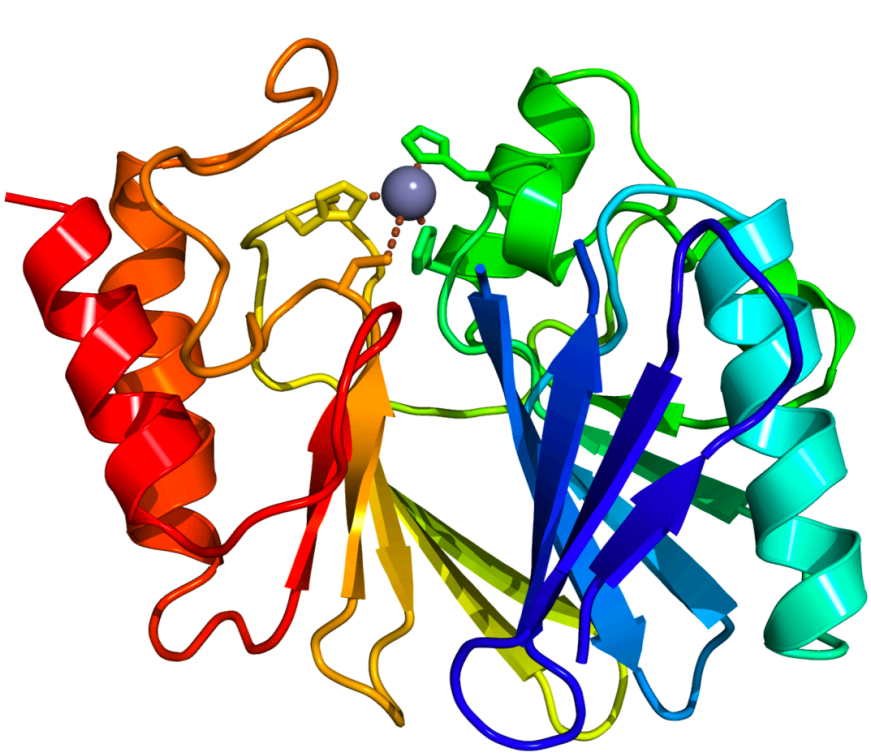
Why we need ever-increasing performance

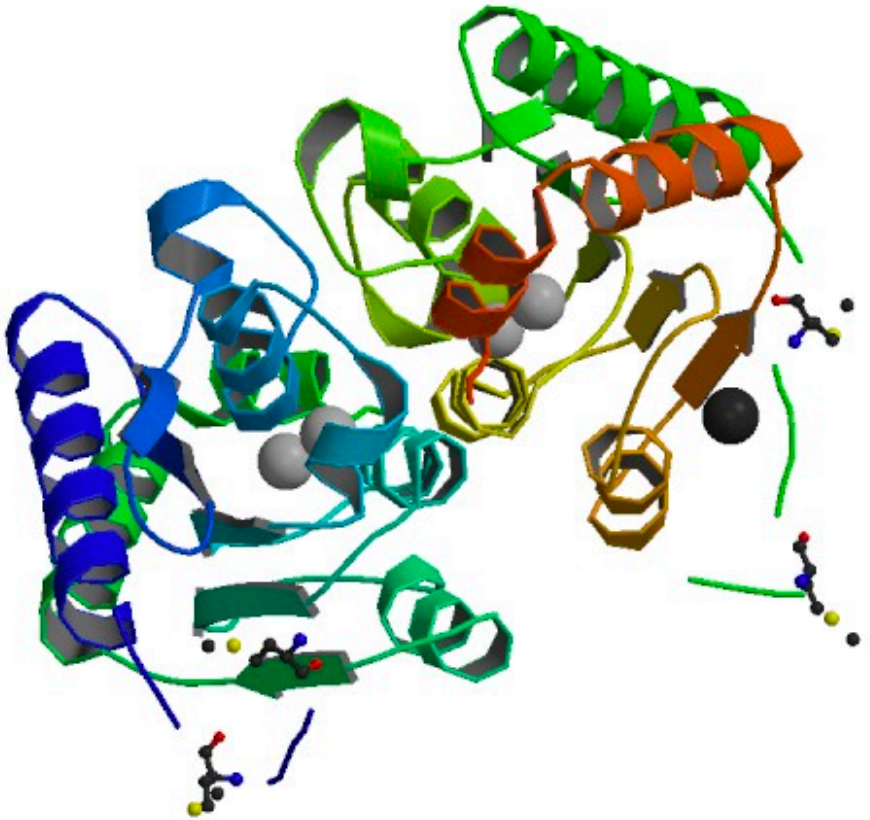
- Computational power is increasing, but so are our computation problems and needs.
- Problems we never dreamed of have been solved because of past increases, such as decoding the human genome.
- More complex problems are still waiting to be solved.

Climate modeling



Protein folding





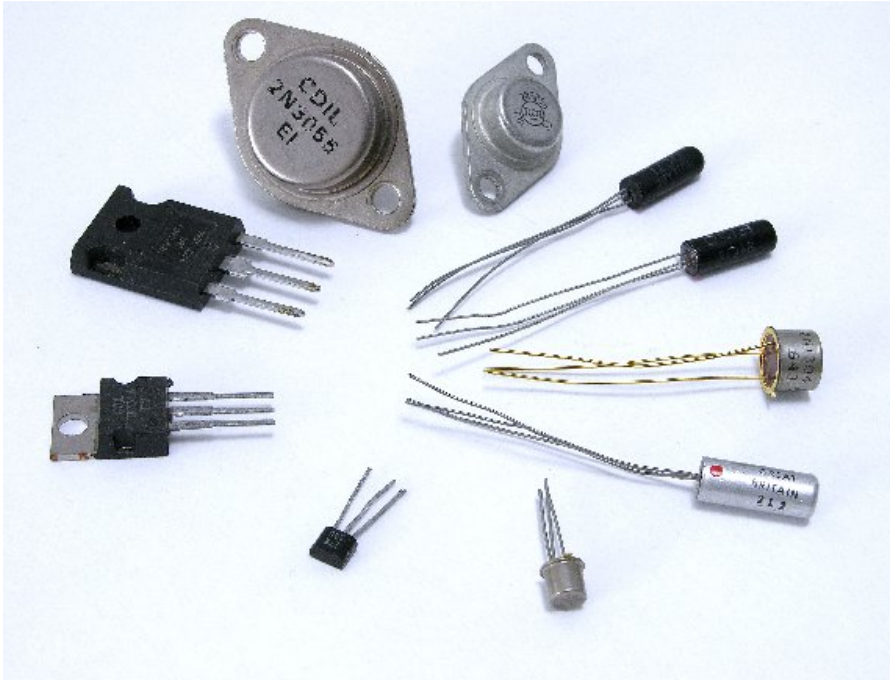
Drug discovery

Energy research

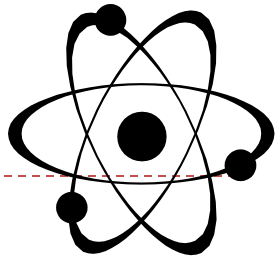
Data analysis

Why we're building parallel systems

- Up to now, performance increases have been attributable to increasing density of transistors.
- But there are inherent problems.



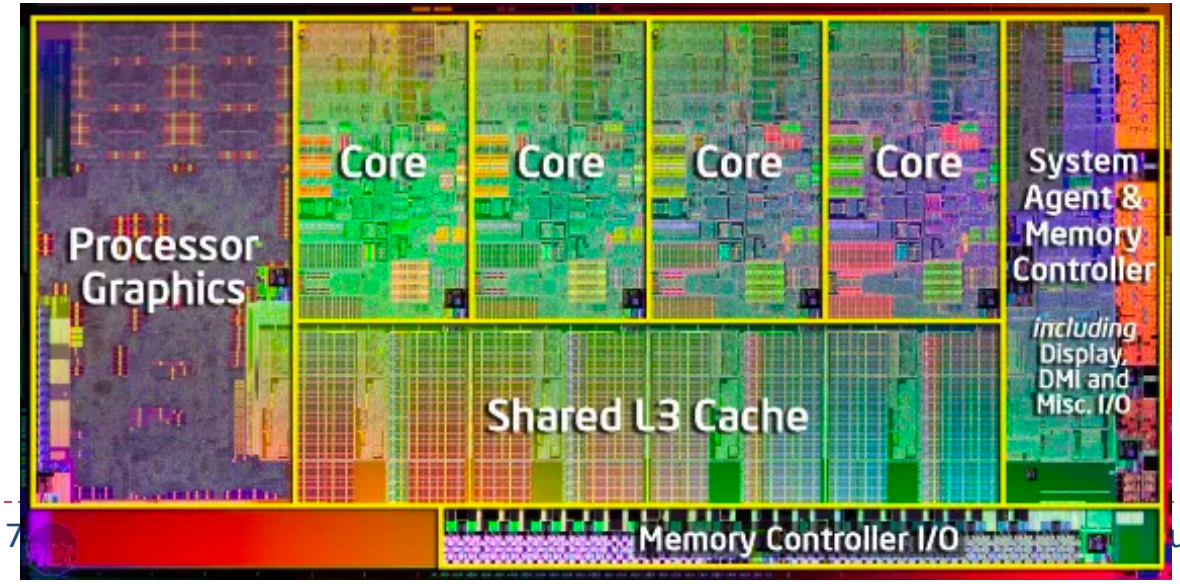
A little physics lesson



- Smaller transistors = faster processors.
- Faster processors = increased power consumption.
- Increased power consumption = increased heat.
- Increased heat = unreliable processors.

Solution

- Move away from single-core systems to multicore processors.
- “core” = central processing unit (CPU)
- Introducing parallelism!!!



Why we need to write parallel programs

- Running multiple instances of a serial program often isn't very useful.
- Rewrite serial programs so that they're parallel.
- Write translation programs that automatically convert serial programs into parallel programs.
  - This is very difficult to do.
  - Success has been limited.

More problems

- Some coding constructs can be recognized by an automatic program generator and converted to a parallel construct.
- However, it's likely that the result will be a very inefficient program.
- Sometimes the best parallel solution is to step back and devise an entirely new algorithm.

Example

- Compute n values and add them together.
- Serial solution:

```
sum = 0;
for (i = 0; i < n; i++) {
    x = Compute_next_value(. . .);
    sum += x;
}
```

Example (cont.)

- We have p cores, p much smaller than n.
- Each core performs a partial sum of approximately n/p values.

```
my_sum = 0;
my_first_i = . . . ;
my_last_i = . . . ;
for (my_i = my_first_i; my_i < my_last_i; my_i++) {
   my_x = Compute_next_value( . . .);
   my_sum += my_x;
}
```

Each core uses it's own private variables and executes this block of code independently of the other cores.

Example (cont.)

- After each core completes execution of the code, is a private variable my_sum contains the sum of the values computed by its calls to Compute_next_value.

- Ex., 8 cores, n = 24, then the calls to Compute_next_value return:

1,4,3, 9,2,8, 5,1,1, 5,2,7, 2,5,0, 4,1,8, 6,5,1, 2,3,9

Example (cont.)

- Once all the cores are done computing their private my_sum, they form a global sum by sending results to a designated “master” core which adds the final result.

Example (cont.)

```
if (I'm the master core) {
    sum = my_x;
    for each core other than myself {
        receive value from core;
        sum += value;
    }
} else {
    send my_x to the master;
}
```

Example (cont.)

1,4,3, 9,2,8, 5,1,1, 5,2,7, 2,5,0, 4,1,8, 6,5,1, 2,3,9

Core	0	1	2	3	4	5	6	7
my_sum	8	19	7	15	7	13	12	14

Global sum

8 + 19 + 7 + 15 + 7 + 13 + 12 + 14 = 95

Core	0	1	2	3	4	5	6	7
my_sum	95	19	7	15	7	13	12	14

But wait!

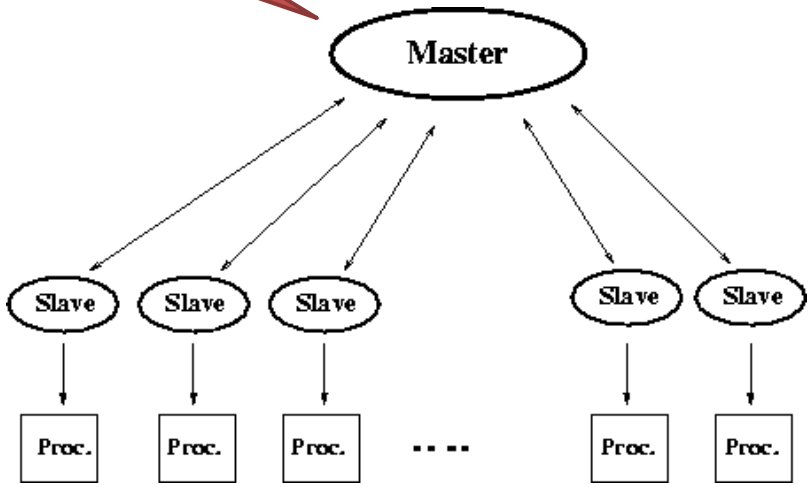
- There's a much better way to compute the global sum.

Problems?

Potential bottleneck!

Global sum

8 + 19 + 7 + 15 + 7 + 13 + 12 + 14 = 95



Core	0	1	2	3	4	5	6	7
my_sum	8	19	7	15	7	13	12	14

1,4,3, 9,2,8, 5,1,1, 5,2,7, 2,5,0, 4,1,8, 6,5,1, 2,3,9

Better parallel algorithm

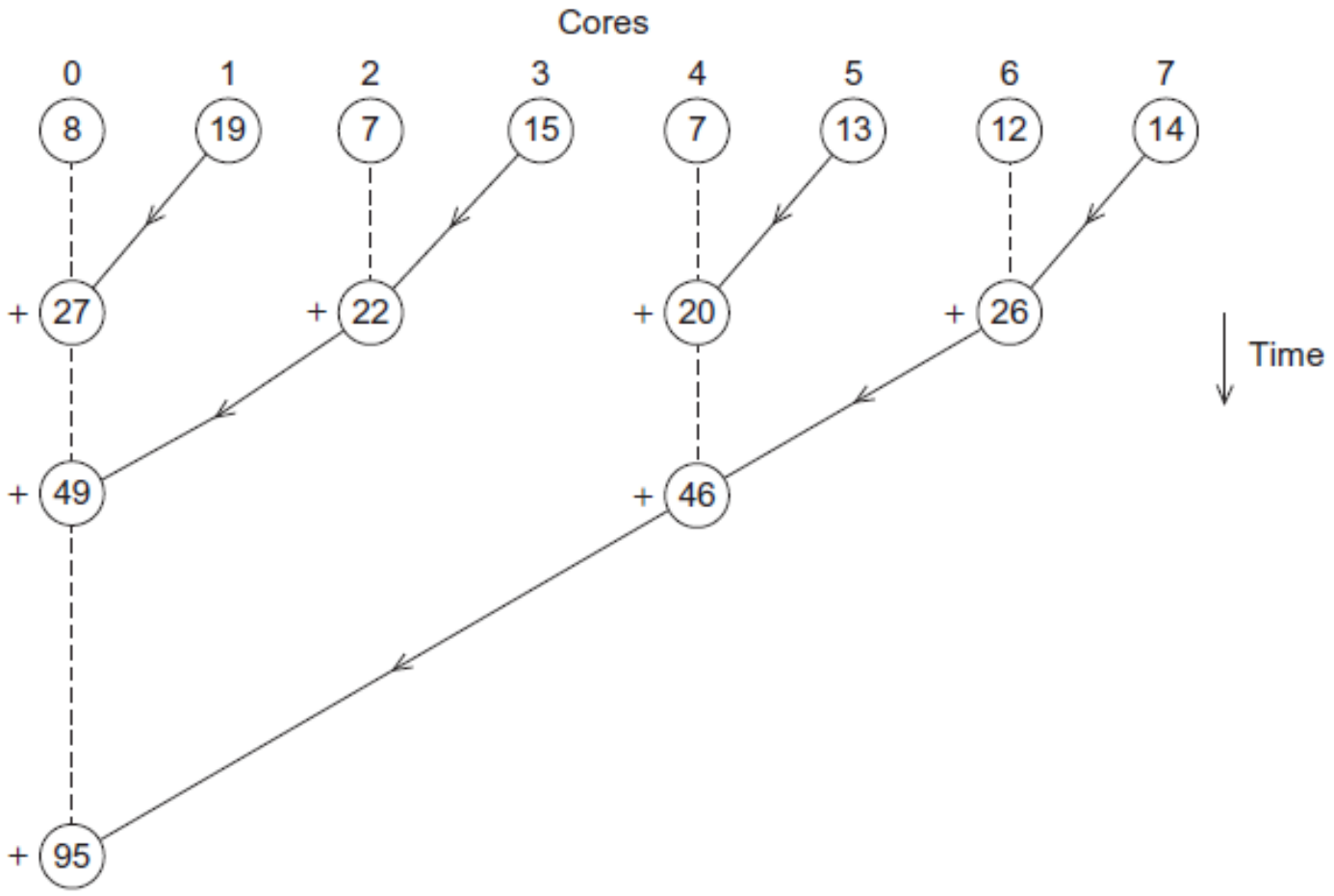
- Don’t make the master core do all the work.
- Share it among the other cores.
- Pair the cores so that core 0 adds its result with core 1’s result.
- Core 2 adds its result with core 3’s result, etc.
- Work with odd and even numbered pairs of cores.

Better parallel algorithm (cont.)

- Repeat the process now with only the evenly ranked cores.
- Core 0 adds result from core 2.
- Core 4 adds the result from core 6, etc.

- Now cores divisible by 4 repeat the process, and so forth, until core 0 has the final result.

Multiple cores forming a global sum



Analysis

- In the first example, the master core performs 7 receives and 7 additions.

- In the second example, the master core performs 3 receives and 3 additions.

- The improvement is more than a factor of 2!

Analysis (cont.)

- The difference is more dramatic with a larger number of cores.
- If we have 1000 cores:
  - The first example would require the master to perform 999 receives and 999 additions.
  - The second example would only require 10 receives and 10 additions.
- That's an improvement of almost a factor of 100!

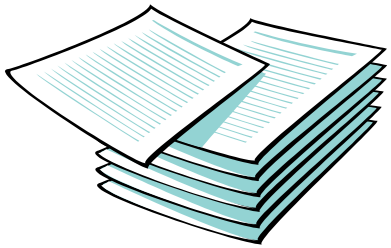
How do we write parallel programs?

- Task parallelism
  - Partition various tasks carried out solving the problem among the cores.
- Data parallelism
  - Partition the data used in solving the problem among the cores.
  - Each core carries out similar operations on it's part of the data.

Professor P

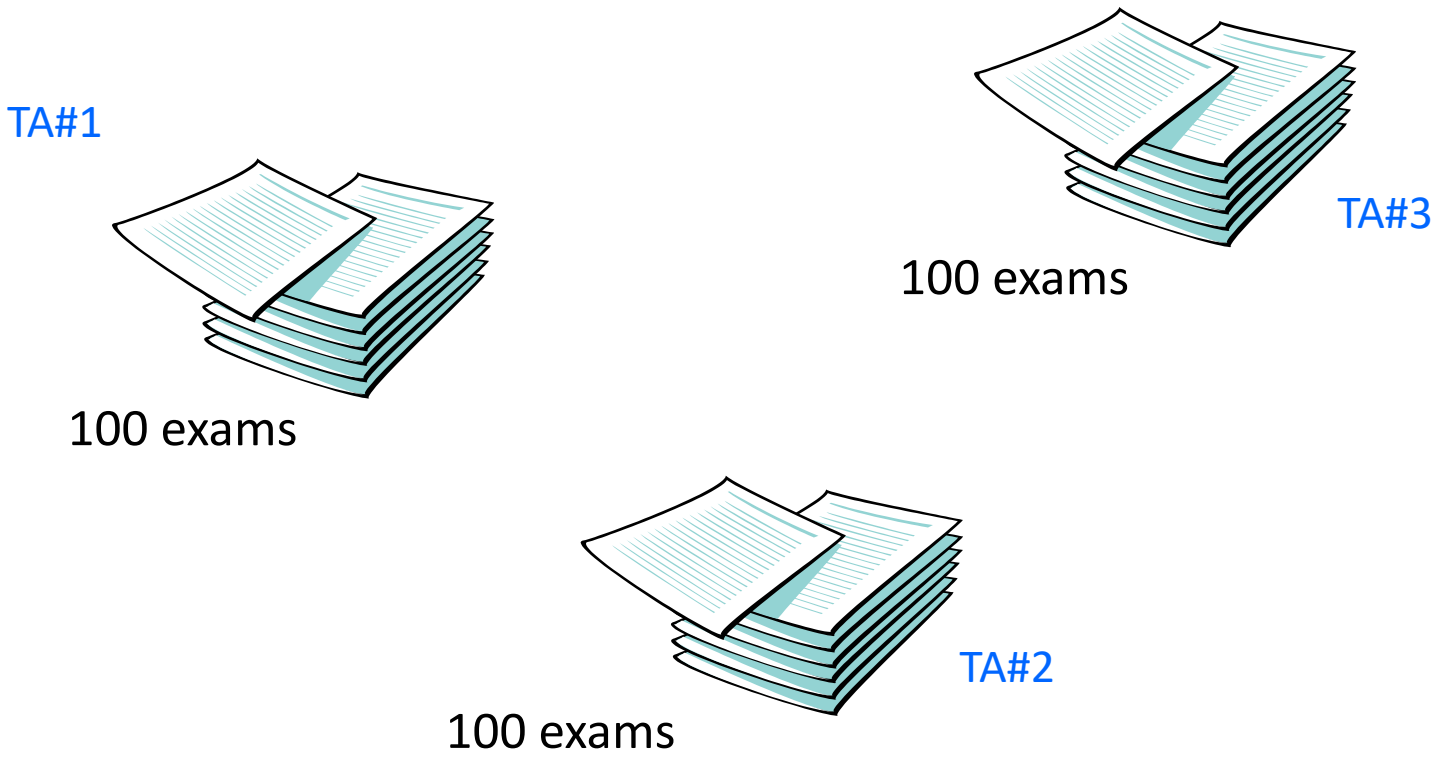
15 questions

300 exams



Professor P's grading assistants

Division of work – data parallelism



Division of work – task parallelism

TA#1

Questions 1 - 5

TA#3

Questions 11 - 15

TA#2

Questions 6 - 10

Division of work – data parallelism

```
sum = 0;
for (i = 0; i < n; i++) {
    x = Compute_next_value(. . .);
    sum += x;
}
```

Same task (program), different data

Division of work – task parallelism

```
if (I'm the master core) {
    sum = my_x;
    for each core other than myself {
        receive value from core;
        sum += value;
    }
} else {
    send my_x to the master;
}
```

different tasks (program): program A and B could be totally different logic

different data

Tasks

1) Receiving
2) Addition

Coordination

- Cores usually need to coordinate their work.
- Communication – one or more cores send their current partial sums to another core.
- Load balancing – share the work evenly among the cores so that one is not heavily loaded.
- Synchronization – because each core works at its own pace, make sure cores do not get too far ahead of the rest.

What we'll be doing

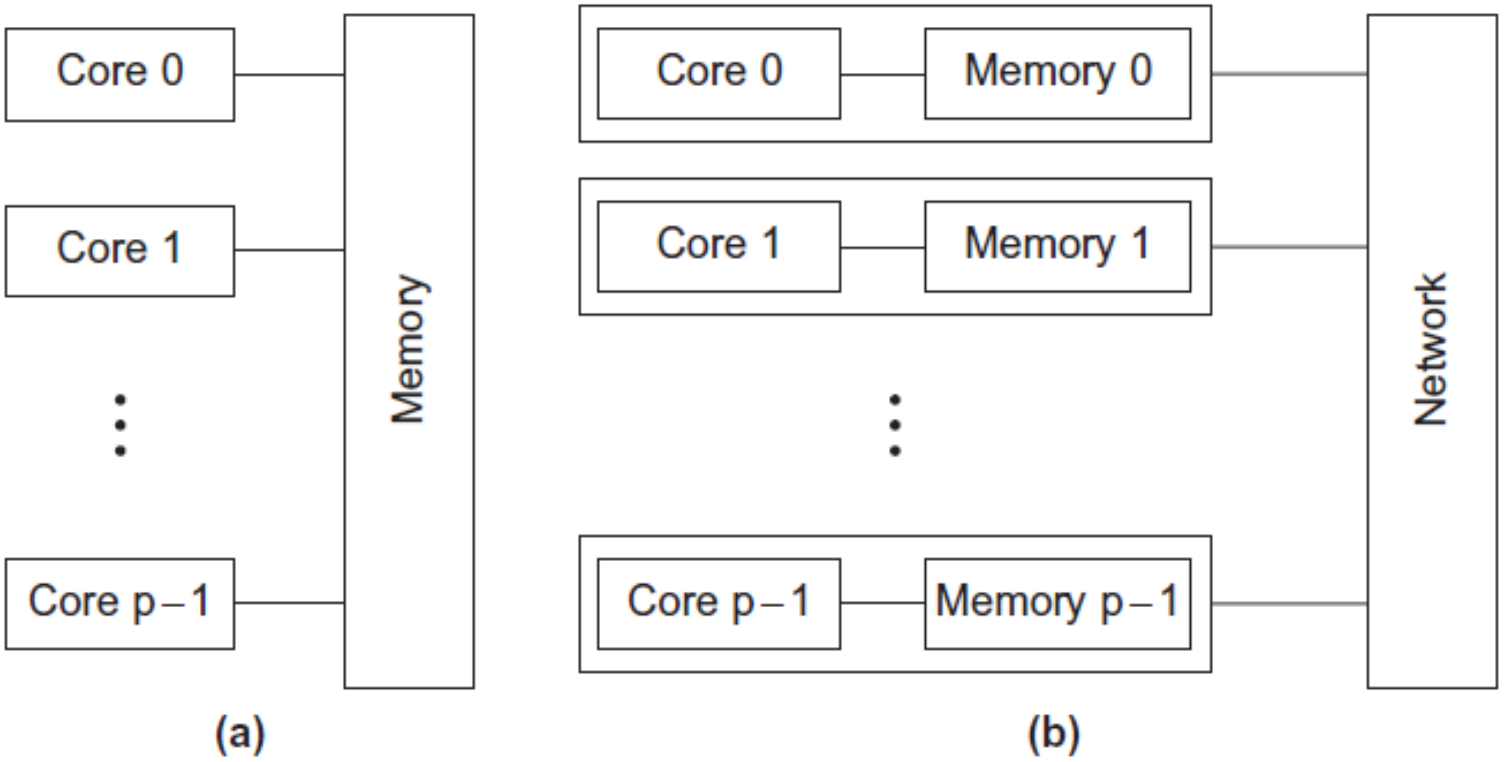
- Using the C language.
- Using three different extensions to C.
  - Message-Passing Interface (MPI)
  - Posix Threads (Pthreads)
  - OpenMP
- Write programs that are explicitly parallel.

Type of parallel systems

- Shared-memory
  - The cores can share access to the computer's memory.
  - Coordinate the cores by having them examine and update shared memory locations.

- Distributed-memory
  - Each core has its own, private memory.
  - The cores must communicate explicitly by sending messages across a network.

Type of parallel systems



Shared-memory

Distributed-memory

Terminology

- Concurrent computing – a program is one in which multiple tasks can be in progress at any instant.

- Parallel computing – a program is one in which multiple tasks cooperate closely to solve a problem

- Distributed computing – a program may need to cooperate with other programs to solve a problem.

Conclusion

- The laws of physics have brought us to the doorstep of multicore technology.
- Serial programs typically don't benefit from multiple cores.
- Automatic parallel program generation from serial program code isn't the most efficient approach to get high performance from multicore computers.

Conclusion (cont.)

- Learning to write parallel programs involves learning how to coordinate the cores.

- Parallel programs are usually very complex and therefore, require sound program techniques and development.