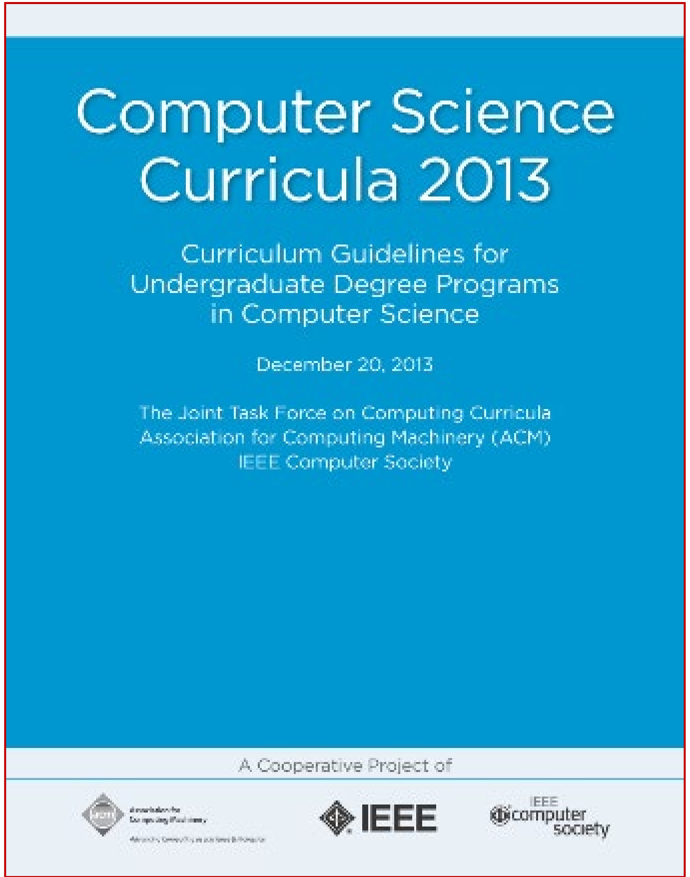
Parallel Programming: Principle and Practice

Course Goals

- The students will get the skills to use some of the best existing parallel programming tools, and be exposed to a number of open research questions
- This course will
  - provide an introduction to parallel computing including parallel computer architectures, analytical modeling of parallel programs, the principles of parallel algorithm design
  - include material on SPMD, OpenMP, MPI, CUDA, *MapReduce, *Tensorflow

IEEE/ACM Computer Science Curricula 2013



PD. Parallel and Distributed Computing (5 Core-Tie

	Core-Tier1 hours
PD/Parallelism Fundamentals	2
PD/Parallel Decomposition	1
PD/Communication and Coordination	1
PD/Parallel Algorithms, Analysis, and Programming	
PD/Parallel Architecture	1
PD/Parallel Performance	
PD/Distributed Systems	
PD/Cloud Computing	
PD/Formal Models and Semantics	

ACM/IEEE-CS CS2013 Course-Exemplar

Syllabus

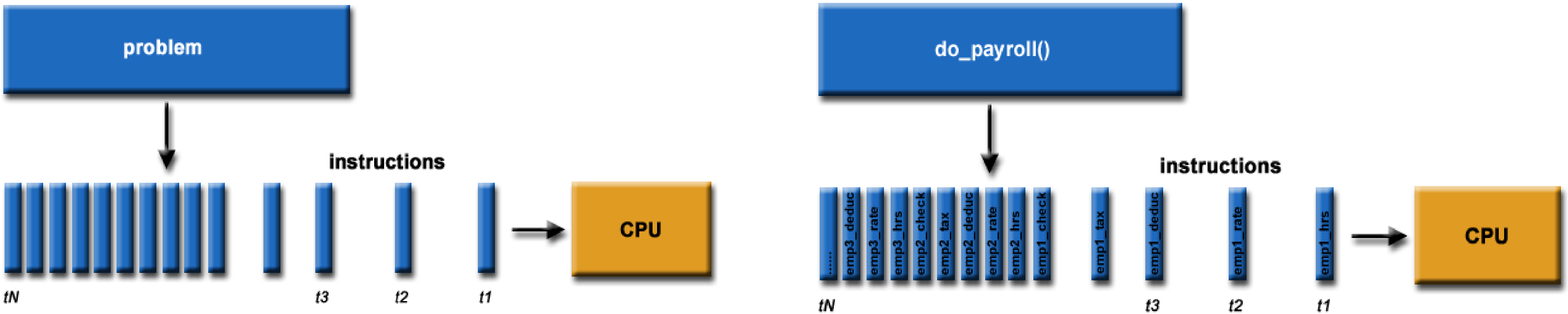
- Part 1: Computer system architecture
  - Single processor system
  - Multicore processors/ multiprocessor with shared memory (SMP)）
  - Computer cluster/multi-computers with distributed memory
  - Accelerator / GPU
  - *New development
- Part 2: Parallel algorithm design and performance analysis
  - Logical parallel programming models
  - Parallel algorithm design
  - Performance analysis
- Part 3: Parallel programming
  - SPMD (single program multiple data)
  - OpenMP
  - MPI
  - CUDA
  - * MapReduce / * Tensorflow

INTRODUCTION

What is Parallel Computing?

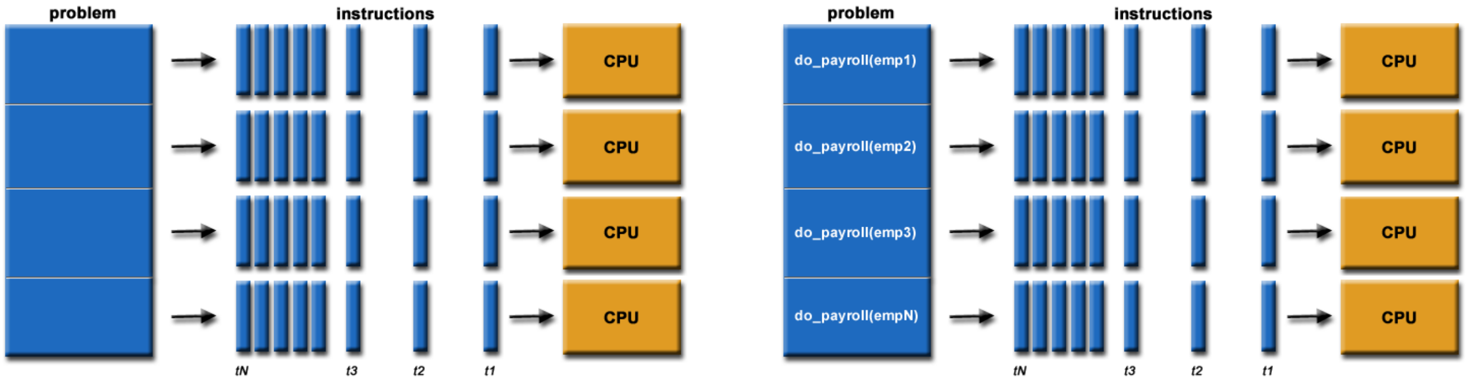
What is Parallel Computing?

- ***Traditionally***, software has been written for serial computation
  - To be run on a single computer having a single CPU
  - A problem is broken into a discrete series of instructions
  - Instructions are executed one after another
  - Only one instruction may execute at any moment in time



Parallel Computing

- Parallel computing is a type of computation in which many calculations or processes are carried out simultaneously.
  - Large problems can often be divided into smaller ones, which can then be solved at the same time

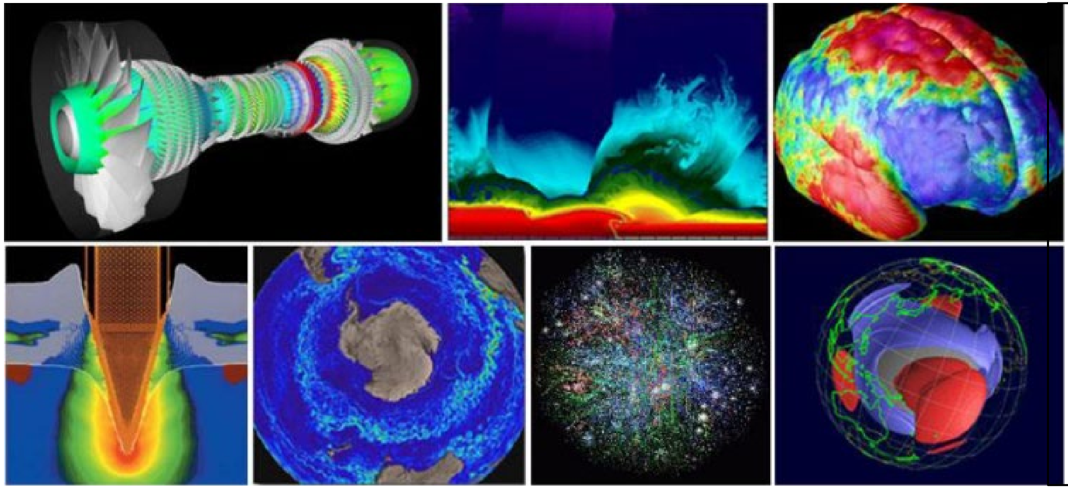


  - different forms: bit-level, instruction-level, data, and task parallelism

source:https://en.wikipedia.org/

Example

- Historically, high-performance computing
- Recent, power consumption (and consequently heat generation)
- Parallel computing has become **the dominant paradigm** in computer architecture, mainly in the form of multi-core processors



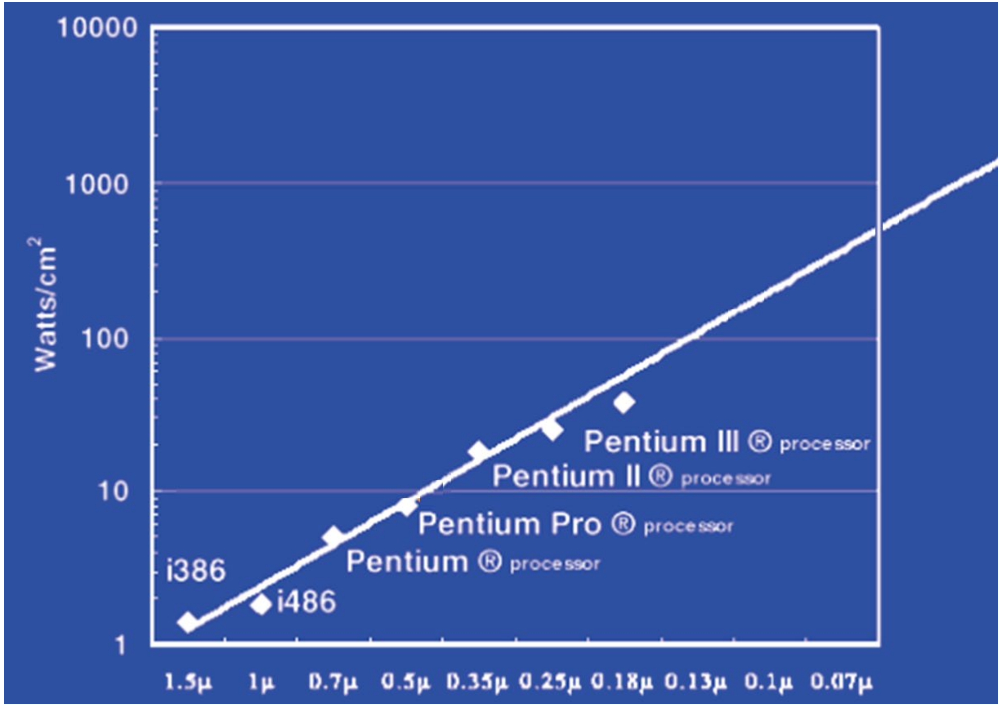
- Physical applications, Nuclear energy, Atomic energy,
- Bioscience, Bioengineering, Genetics
- Chemistry, Molecular Science
- Geography and Seismology
- Mechanical engineering, from prosthetics to space vehicles
- Electrical engineering, circuit design, microelectronics

INTRODUCTION

Why Parallel Computing?

High Performance

- Speed
  - clock rate/frequency
- High speed
  - High speed computing means getting a particular job done in less time (e.g., calculations per second)



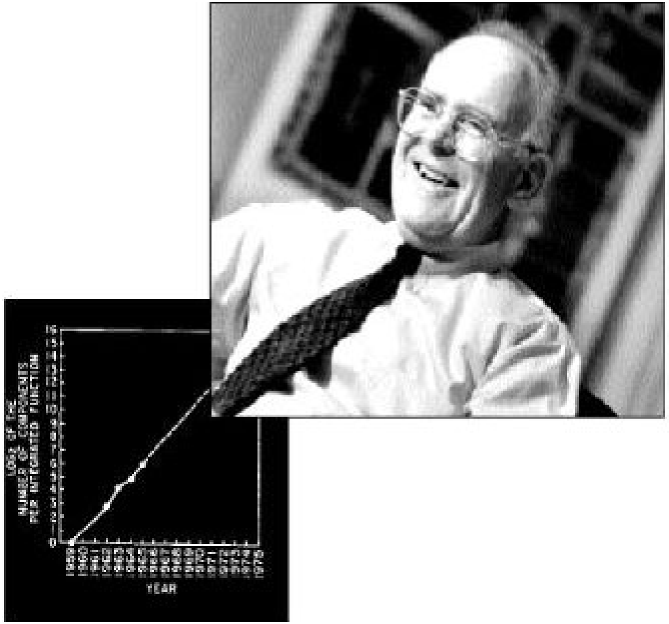
High Performance

- Throughput
  - the amount of work completed in a unit of time
  - the processes executed to number of jobs completed in a unit of time
- High throughput
  - high throughput computing means getting lots of work done per large time unit (e.g., jobs per month)

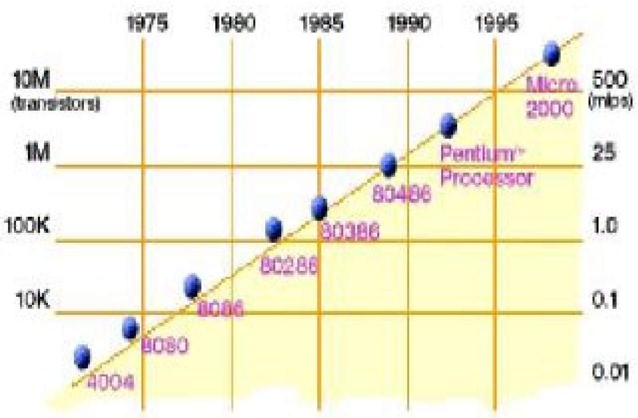
INTRODUCTION

Technology Push

Technology push: Moore's Law



2X transistors/Chip Every 1.5 years



Called "Moore's Law"

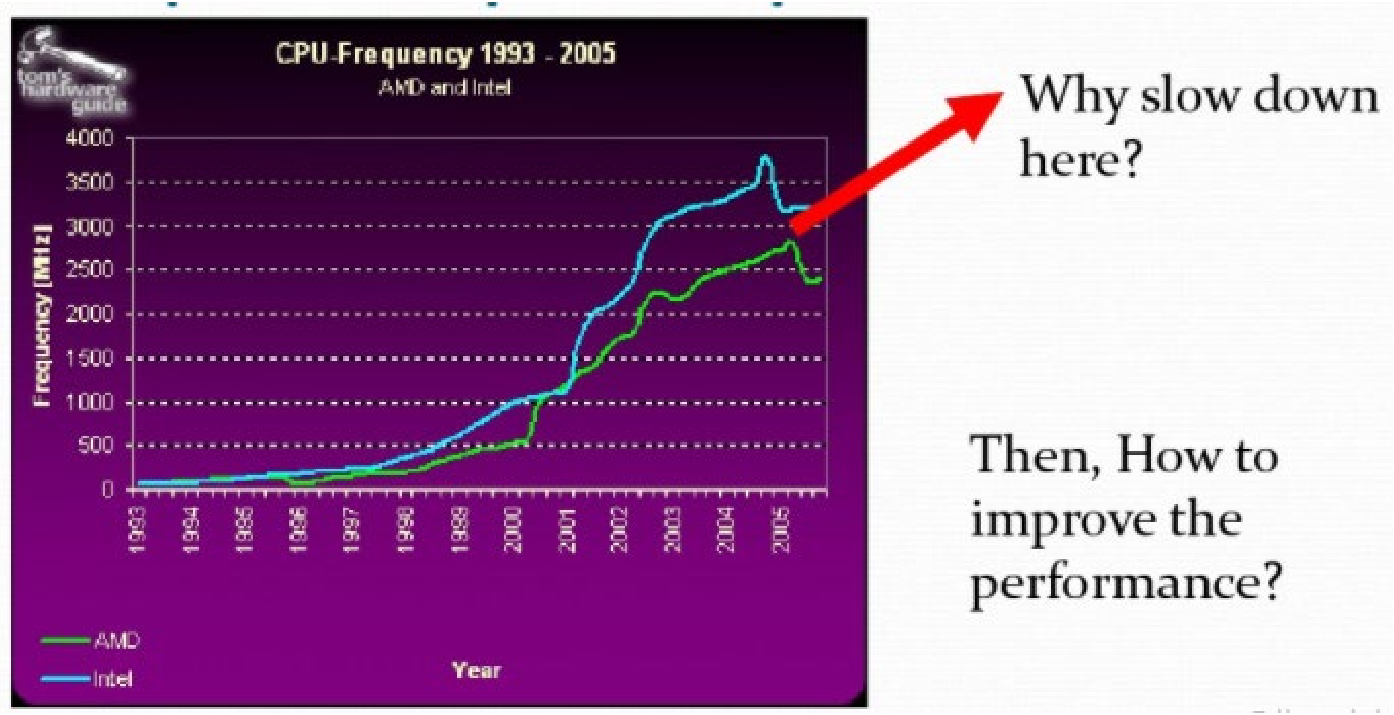
Gordon Moore (co-founder of Intel) predicted in 1965 that the transistor density of semiconductor chips would double roughly every 18 months.

Microprocessors have become smaller, denser, and more powerful.

Technology push: Frequency/Power wall

- Power = activity * capacitance * voltage^2 * frequency
- Frequency/Power wall
  - It is not possible to consistently run at higher frequencies without hitting power/thermal limits
  - Power wall makes it harder to add complex features
  - Power wall makes it harder to increase frequency

Technology push: Frequency/Power wall



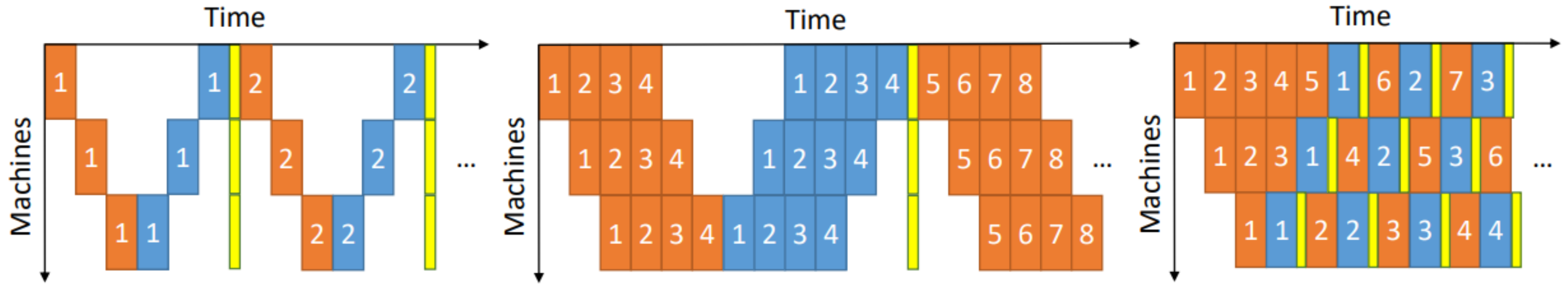
The development tendency is not higher clock rates

Technology push: ILP wall

- ILP: Instruction Level Parallelism
  - ILP allows the compiler and the processor to overlap the execution of multiple instructions or even to change the order in which instructions are executed
- Examples of ILP techniques
  - Pipelining: Overlapping individual parts of instructions
  - Superscalar execution: Do multiple things at same time
  - VLIW: Let compiler specify which operations can run in parallel
  - Vector Processing: Specify groups of similar (independent) operations
  - Out of Order Execution (OOO): Allow long operations to happen

Technology push: ILP wall

- Gantt charts comparing pipelined synchronous model parallelism and asynchronous model parallelism
  - The numbers in the boxes indicate instance IDs
  - Orange, blue, and yellow boxes correspond to forward, backward, and parameter update operations, respectively



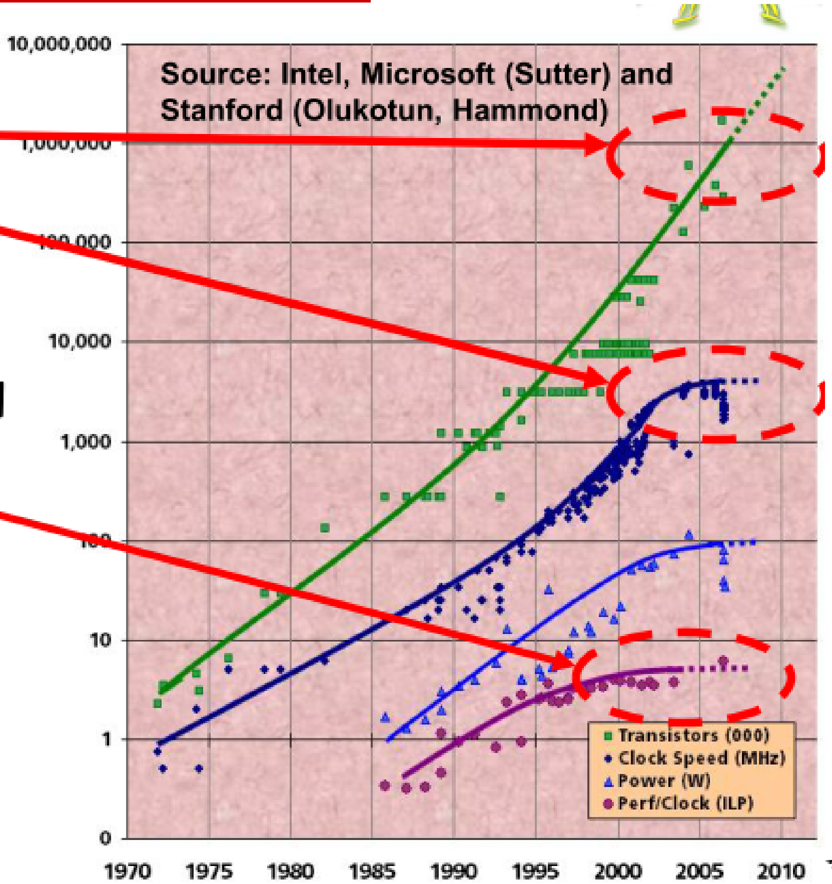
(a) Pipelined execution (#inflight 1) (b) Pipelined parallel execution (#inflight 4) (c) Asynchronous model parallelism

Technology push: ILP wall

- The processor's instruction-level parallelism improvement is approaching its limit
  - Long instruction words, Pipelining, Branch prediction, Register naming, Superscalar, Out-of-order execution, Dynamic issue, Cache…
  - Various complex micro-architecture technologies such as advanced pipelines have been studied and applied, and it is difficult to further explore more instruction-level parallelism

Limiting Forces: Clock Speed and ILP

- Chip density is continuing increase ~2x every 2 years
- Clock speed is not
  - # processors/chip (cores) may double instead
- There is little or no more Instruction Level Parallelism (ILP) to be found
  - Can no longer allow programmer to think in terms of a serial programming model
- Conclusion: Parallelism must be exposed to software!



Multicore processor(Recent Intel Processors)

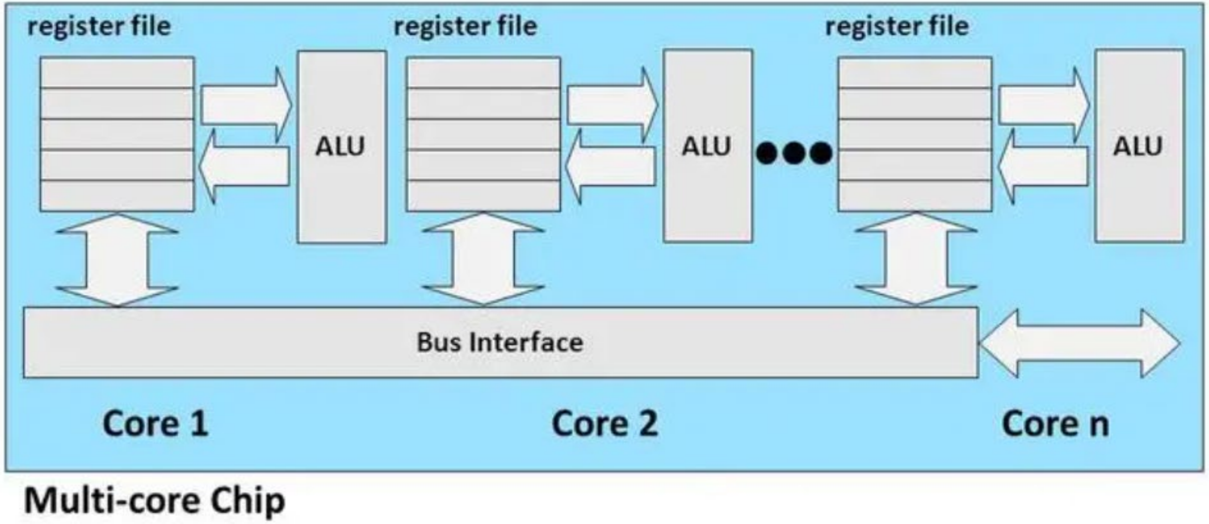
Copyright © Intel

Processors	Year	Fabrication(nm)	Clock(GHz)	Power(W)
Pentium 4	2000	180	1.80-4.00	35-115
Pentium M	2003	90/130	1.00-2.26	5-27
Core 2 Duo	2006	65	2.60-2.90	10-65
Core 2 Quad	2006	65	2.60-2.90	45-105
Core i7(Quad)	2008	45	2.93-3.60	95-130
Core i5(Quad)	2009	45	3.20-3.60	73-95
Pentium Dual-Core	2010	45	2.80-3.33	65-130
Core i3(Duo)	2010	32	2.93-3.33	18-73
2nd Gen i3(Duo)	2011	32	2.50-3.40	35-65
2nd Gen i5(Quad)	2011	32	3.10-3.80	45-95
2nd Gen i7(Quad/Hexa)	2011	32	3.80-3.90	65-130
3rd Gen i3(Duo)	2012	22/32	2.80-3.40	35-55
3rd Gen i5(Quad)	2012	22/32	3.20-3.80	35-77
3rd Gen i7(Quad/Hexa)	2012	22/32	3.70-3.90	45-77
Xeon E5(8-cores)	2013	22	1.80-2.90	60-130
Xeon Phi(60-cores)	2013	22	1.10	300

“We are dedicating all of our future product development to multicore designs. We believe this is a key inflection point for the industry.” Intel President Paul Otellini, IDF 2005

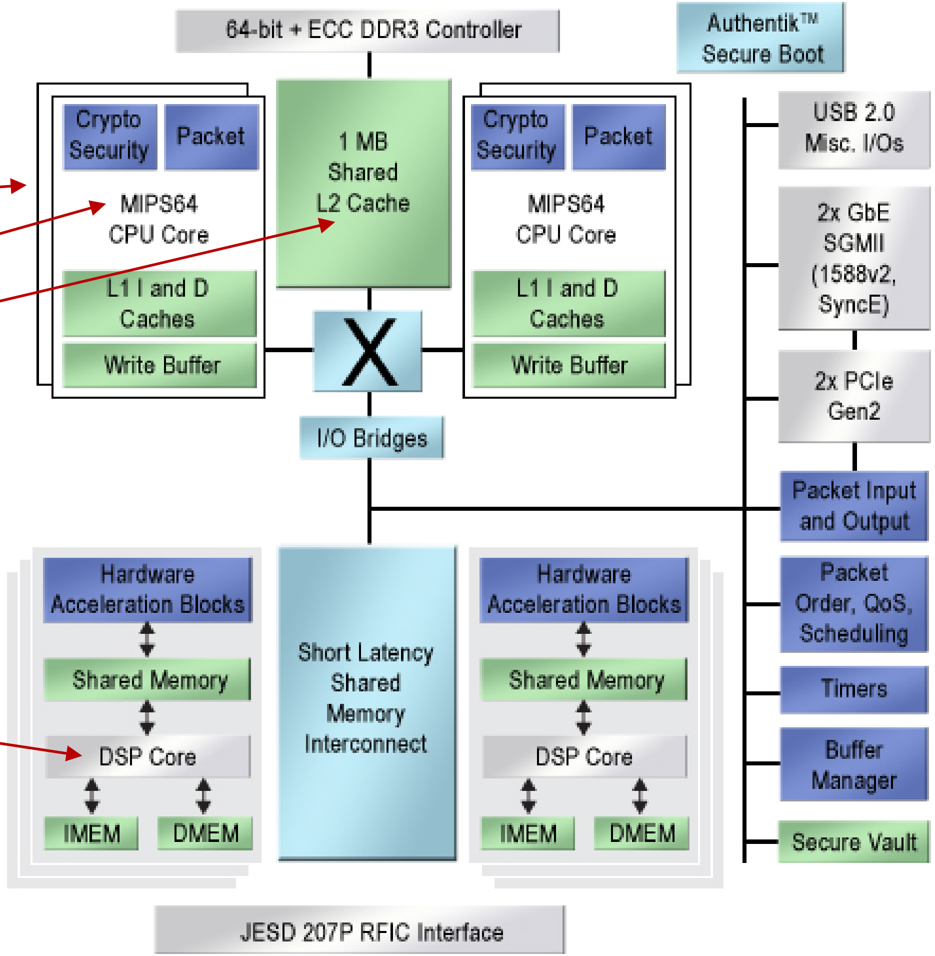
microelectronic technology limitations – multicore processor

- Multi-core architecture
  - Somewhat recent trend in computer architecture
  - Replicate many cores on a single die



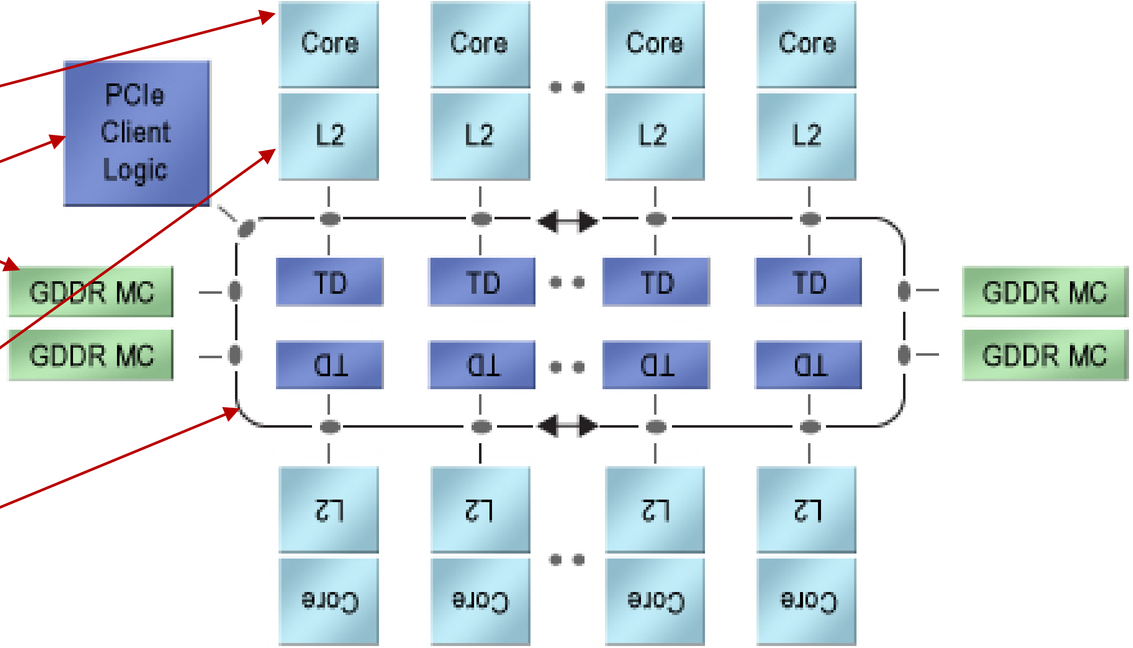
microelectronic technology limitations – multicore processor

- Octeon Fusion CNF71xx
  - 两个处理簇
  - 四个一组的增强MIPS64内核
  - 一个共享L2高速缓存的各种硬件加速器
  - 6个为一组的数字信号处理(DSP)内核
    - 每个内核都有很多硬件加速器
    - 这些内核分布在共享存储器交换架构周围



microelectronic technology limitations – multicore processor

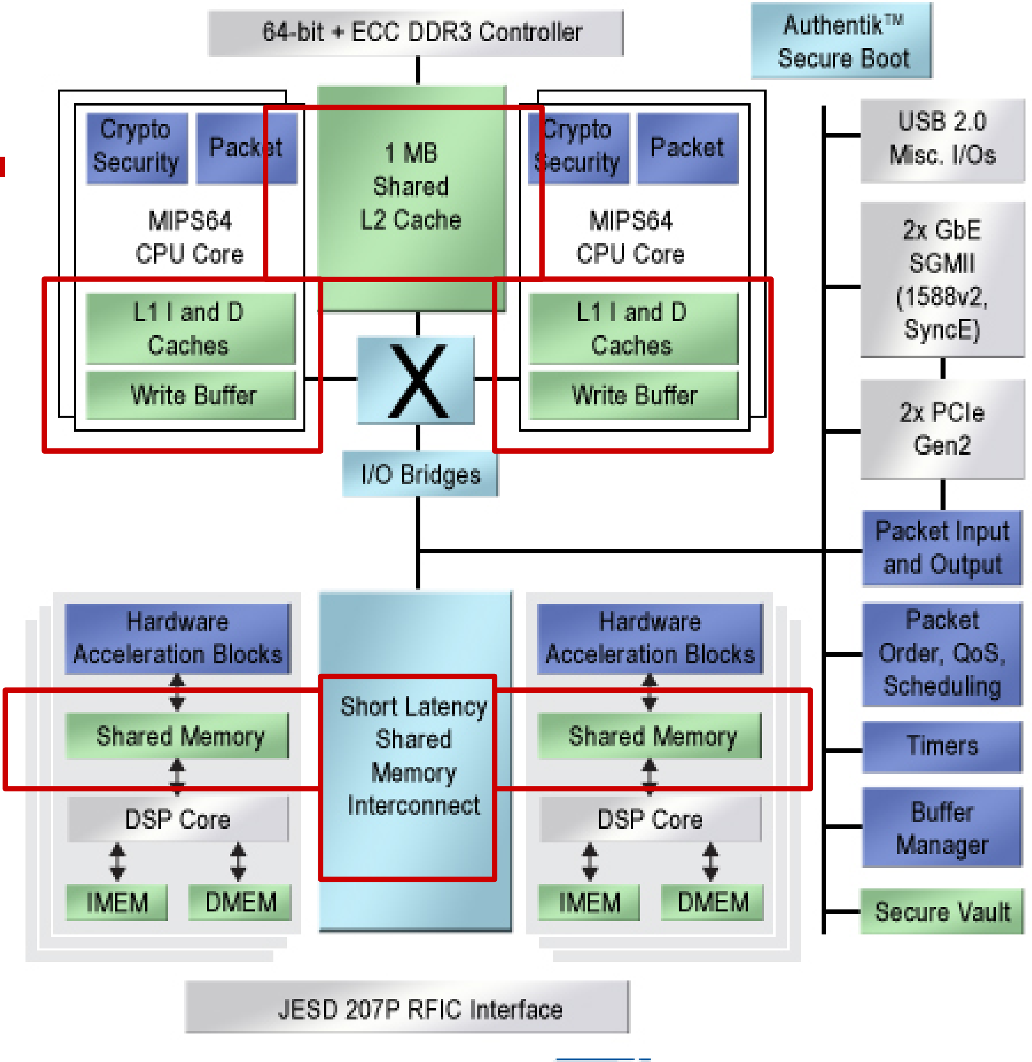
- Intel Xeon Phi: milticore向many core的深入发展
- “50多个”x86处理器内核的协处理器
- 四个GDDR存储器控制器
- 与主处理器Xeon CPU连接的PCI Express® (PCIe®)接口
- 每一个处理器都有自己的专用矢量处理单元，以及自己的512 Kbyte L2高速缓存
  - L2高速缓存、GDDR控制器以及PCIe控制器不是由传统的交换矩阵连接的，这样会导致规模非常大而在物理上无法实现，而是由双向环形总线连接
  - 这一总线在每一方向上都有64字节数据通路，通过分布式标签方案来实现所有L2之间的一致性



microelectronic technology limitations

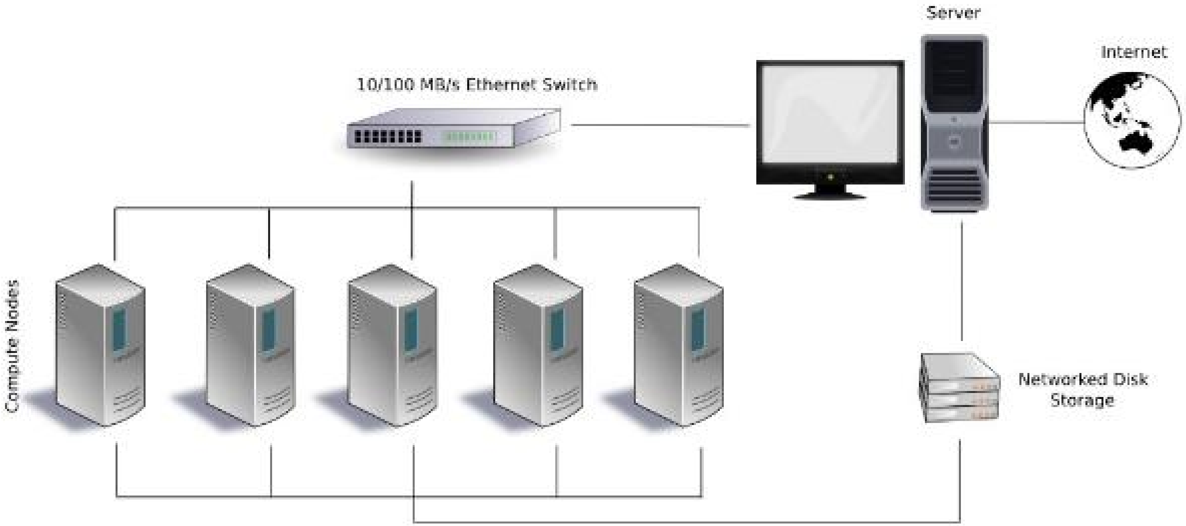
– Shared memory

- simultaneously accessed by multiple programs with an intent to provide communication among them or avoid redundant copies
- an efficient means of passing data between programs



Rapid development of network technology – Computer cluster

- a set of connected computers (nodes) that work together as if they are a single (much more powerful) machine
- range from a simple two-node system connecting two personal computers to a supercomputer with a cluster architecture



Rapid development of network technology – Parallel computers

bullx B700 DLC"*Mistral*"

vendor: Atos/Bull

since July 2015 / July 2016 (installed in 2 phases)

- 36 000 / 100 000 compute cores
- 1.4 / 3.6 PFLOPS peak performance
- 120 / 240 TByte memory
- 54 PByte parallel file system

IBM Power 6 p375 "*Blizzard*"

vendor: IBM

March 2009 - October 2015

- 8 448 compute cores
- 158 TFLOPS peak performance
- 20 TByte memory
- 6 PByte parallel file system

Sun Linux Cluster "*Tornado*"

vendor: Sun Microsystems

April 2008 - July 2012

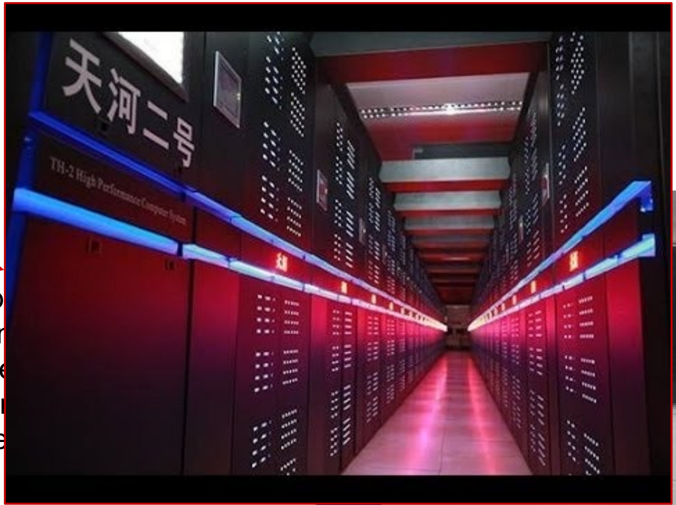
- 256 Sun Fire X2200M- compute nodes
- 2048 compute cores (2 Quad-Core-CPUs per compute node)
- 10 TFLOPS peak performance
- 8.5 TByte memory

Top 500: https://top500.org

TOP 10 Sites for June 2016

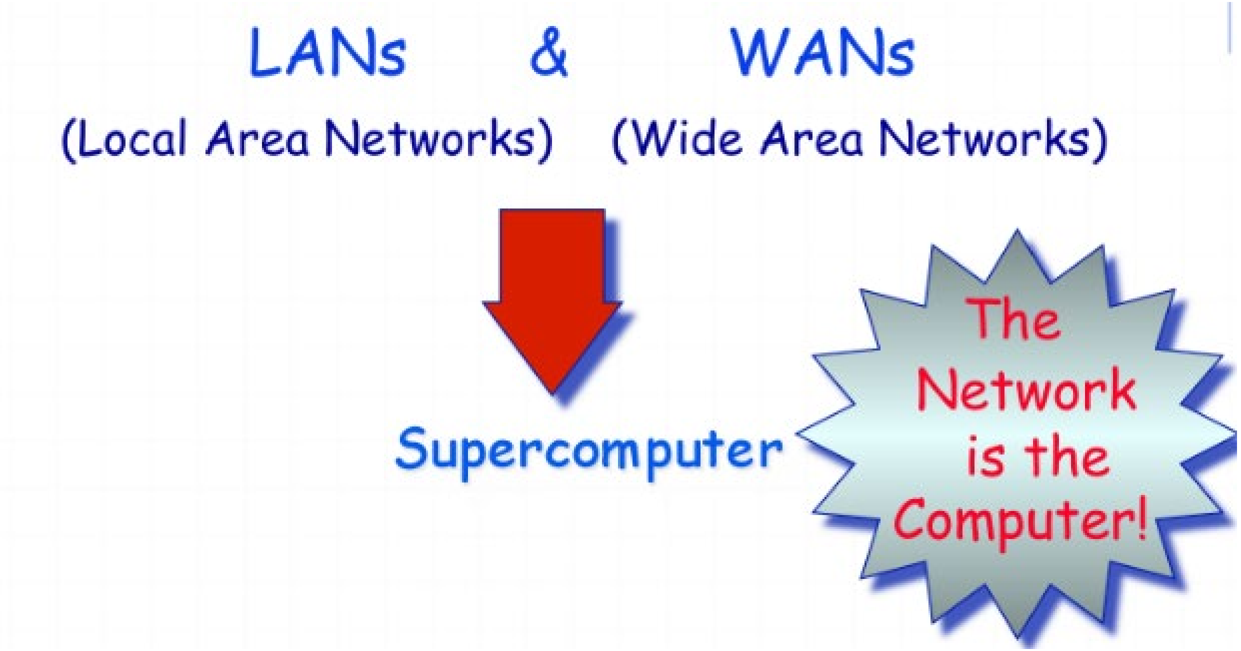
For more information about the sites and systems in the list, click on the links or view the complete list.

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	Sunway TaihuLight - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway, NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371
2	Tianhe-2A - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P, NUDT National Super Computer Center in Guangzhou China	3,120,000	33,862.7	54,902.4	17,808
3	Titan - Cray XK7, Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x, Cray/HPE DOE/SC/Oak Ridge National Laboratory United States	560,640	17,590.0	27,112.5	8,209



Tianhe-2 is a 33.86-petaflops sup
National Supercomputer Center i
developed by a team of 1,300 sci
It was the world's fastest superco
TOP500 lists for June 2013, Nove
November 2014, June 2015, and

Rapid development of network technology – The Network is the Computer



"when the network is as fast as the computer's internal links, the machine disintegrates across the net into a set of special purpose appliances"

Rapid development of network technology – Cloud Computing

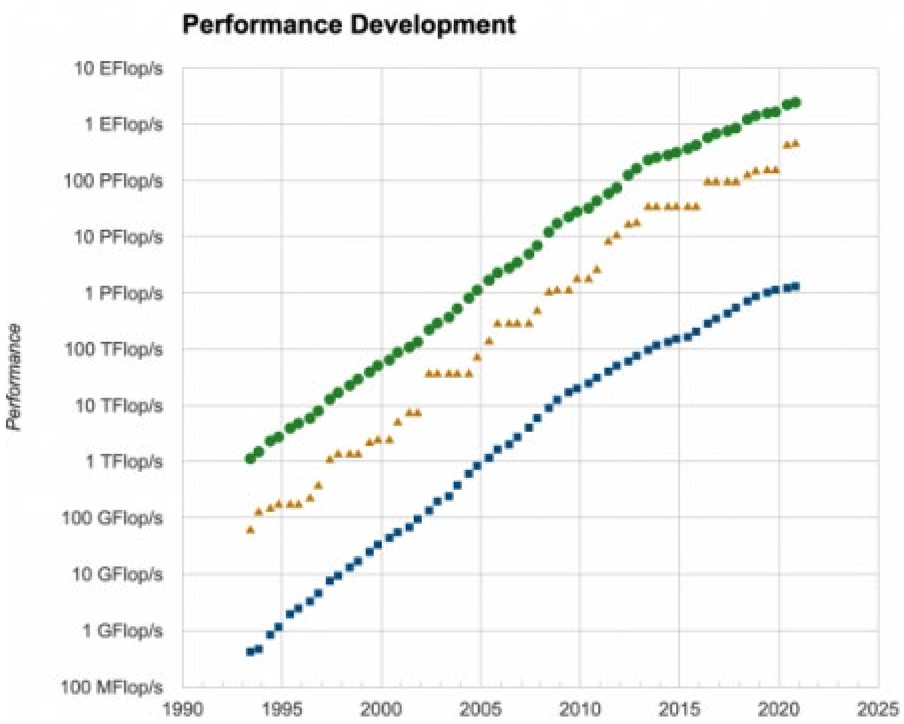
- A style of computing where massively scalable IT-related capabilities are provided “as a service” using **Internet technologies** to multiple external customers
- Cloud computing describes a new supplement, consumption and delivery model for IT services based on the Internet, and it typically involves the provision of dynamically scalable and often **virtualized resources (storage, platform, infrastructure, and software) as a service** over the Internet

All computers are now parallel computers

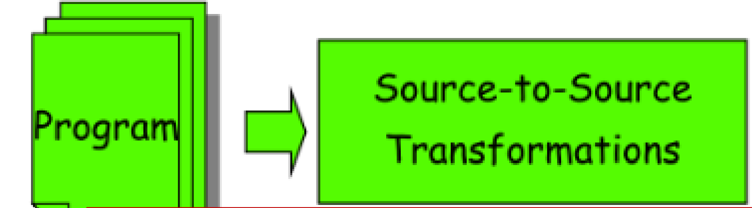
- High-performance computers are currently parallel computers
- The use of parallel computers is increasingly "civilian", and the problems solved by parallel computers are not limited to defense and scientific and technological "cutting-edge" tasks

Trends to Exascale Performance

- During the past 20+ years, the trends indicated by ever faster networks, distributed systems, and multi-processor computer architectures (even at the desktop level) clearly show that ***parallelism is the future of computing***.
- In this same time period, there has been a greater than **500,000x** increase in supercomputer performance, with no end currently in sight.

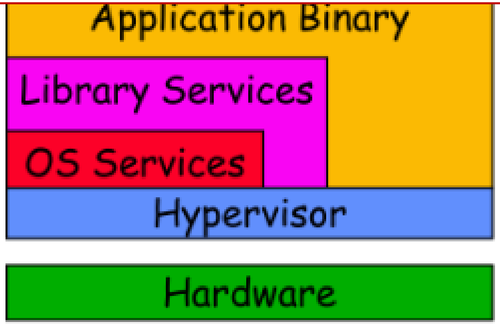


Execution is *not* just about hardware



- The VAX fallacy
  - Produce one instruction for every high-level concept
  - Absurdity: polynomial multiply

> - **Hardware issues now become algorithmic and software problems**
> - **How to effectively and efficiently use the available computer resources for HPC**



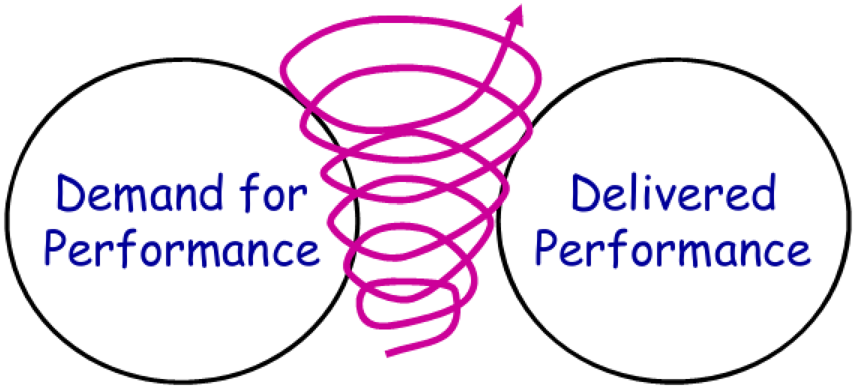
  - Cross-boundary optimization
- Modern programmer does not see assembly language
  - Many do not even see "low-level" languages like "C"

INTRODUCTION

Application Driven

Application Trends

There is a positive feedback cycle between delivered performance and applications' demand for performance



Example application domains:

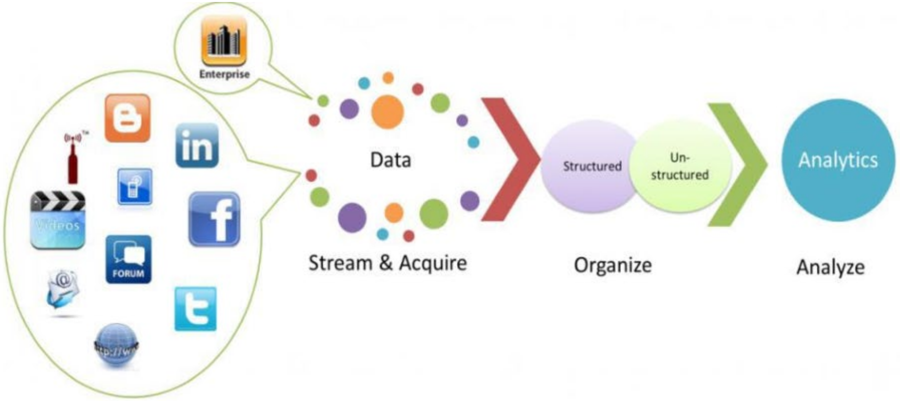
- *Scientific computing* : CFD, Biology, Chemistry, Physics, ...
- *General-purpose computing* : Video, Graphics, CAD, Databases, …

Application driven

- **Incredible Things That Happen Every Minute On The Internet**
  - more than 500 hours of content uploaded on YouTube
  - 695,000 stories shared on Instagram
  - nearly 70 million messages sent via WhatsApp and Facebook Messenger
  - two million swipes on Tinder
  - an incredible 1.6 million U.S. dollars spent online

Application driven

- Big Data Phenomenon
  - The quantitative explosion of digital data has forced researchers to find new ways of seeing and analyzing the world
  - It's about discovering new orders of magnitude for capturing, searching, sharing, storing, analyzing and presenting data

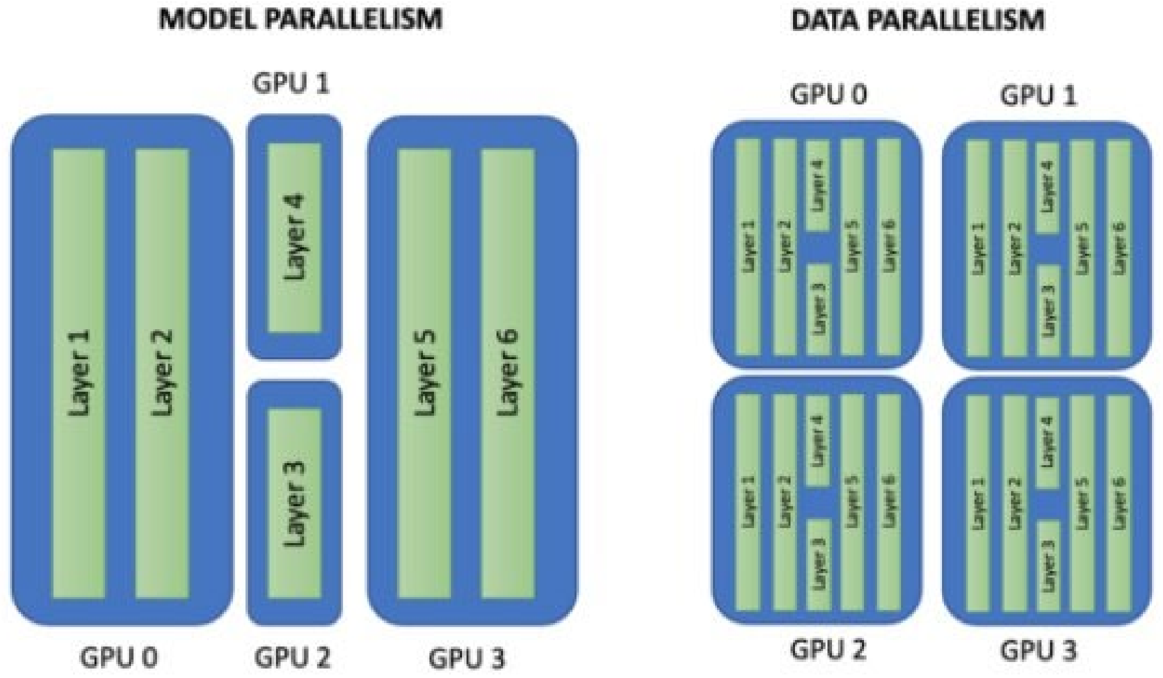


Application driven

- **Parallel computing in AI**
  - Machine learning algorithms have a lot of linear algebra used in them, appearing as matrix vector products.
    - They also have lots of parallel work (many training samples).
  - Neural networks are the best example of connection of AI and parallel computing
    - It is modeled to work like brain, it process information in parallel, learn the patterns and later used to recognize things
    - For example, Google made a huge neural network, gave it a huge set of images and it started recognizing cats.

Application driven

- Parallel computing used in machine learning



INTRODUCTION

Why Study Parallel Computing?

Example

- The computing and algorithm design for parallel computing are more complex than serial algorithm , so we need new ways to learn and design it.

- Here is the 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'd)

- 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 its own private variables and executes this block of code independently of the other cores.

Example (cont'd)

- After each core completes execution of the code, 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'd)

- 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

```
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'd)

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

Better parallel algorithm

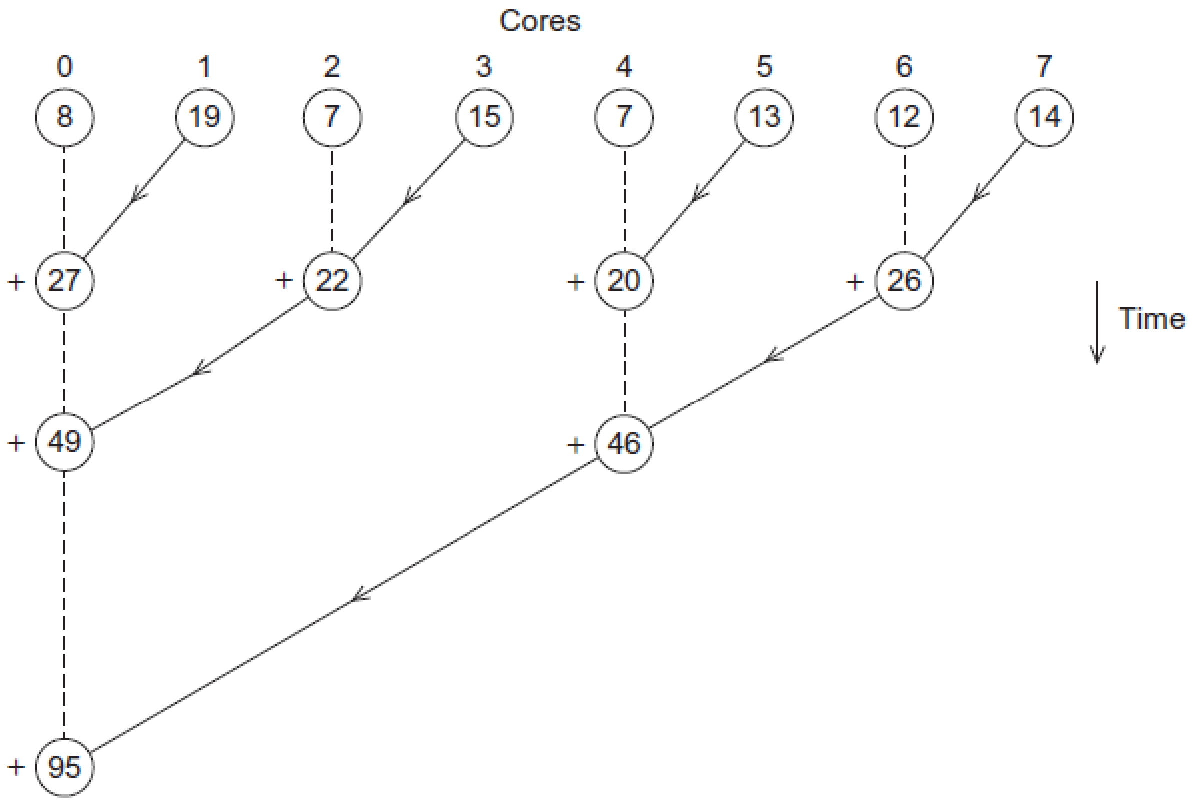
- Don't make the master core do all the work
- Share it among the other cores
- Pair the cores so that core 0 adds its result with core 1's result
- Core 2 adds its result with core 3's result, etc
- Work with odd and even numbered pairs of cores

Better parallel algorithm (cont'd)

- 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'd)

- 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

Parallel Compared to Sequential Programming

- Has different costs, different advantages
- Requires different, unfamiliar algorithms
- Must use different abstractions
- More complex to understand a program's behavior
- More difficult to control the interactions of the program's components
- Knowledge/tools/understanding more primitive

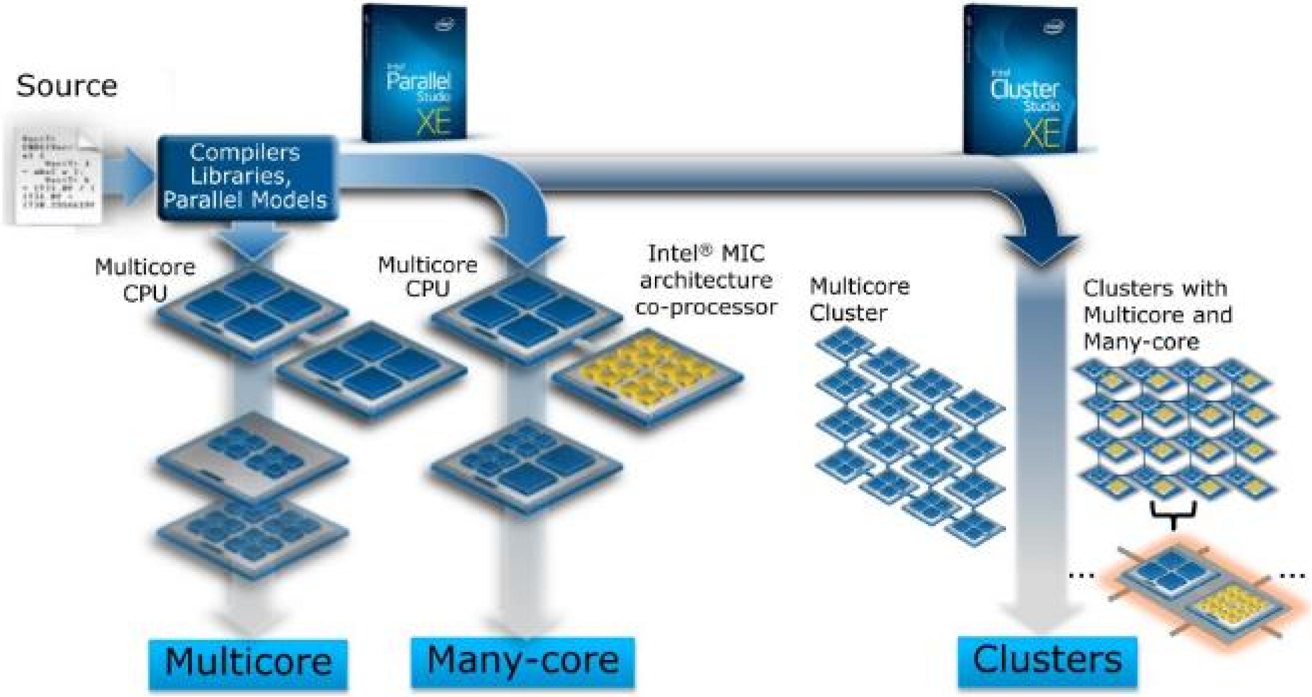
Parallel Programming Complexity

- Enough parallelism? (Amdahl's Law)
- Granularity
- Locality
- Load balance
- Coordination and Synchronization

All of these things makes parallel programming even harder than sequential programming

Parallelizing Compilers

Now we can get: single-source approach to multi- and many-core



Parallelizing Compilers

However:

- After 30 years of intensive research
  - only limited success in parallelism detection and program transformations
    - instruction-level parallelism at the *basic-block level can be detected*
    - parallelism in *nested for-loops containing arrays with simple index expressions can be analyzed*
    - analysis techniques, such as data dependence analysis, pointer analysis, flow sensitive analysis, abstract interpretation, … when applied across procedure boundaries often take far too long and tend to be fragile, i.e., can break down after small changes in the program
  - instead of training compilers to recognize parallelism, ***people have been trained to write programs that parallelize***

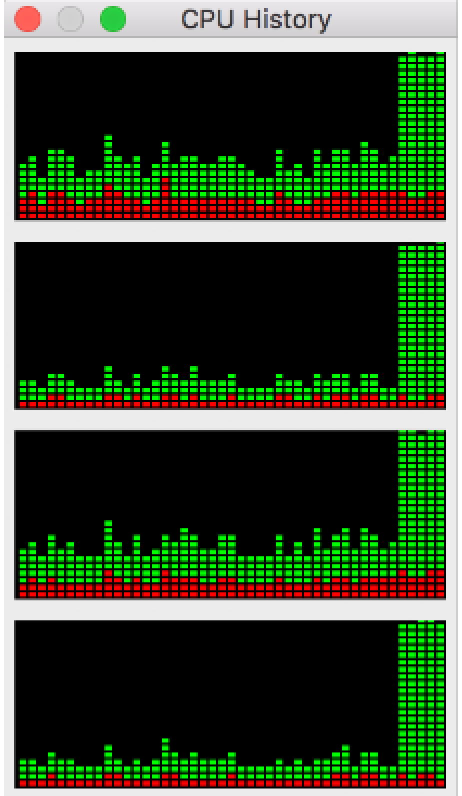
Another example

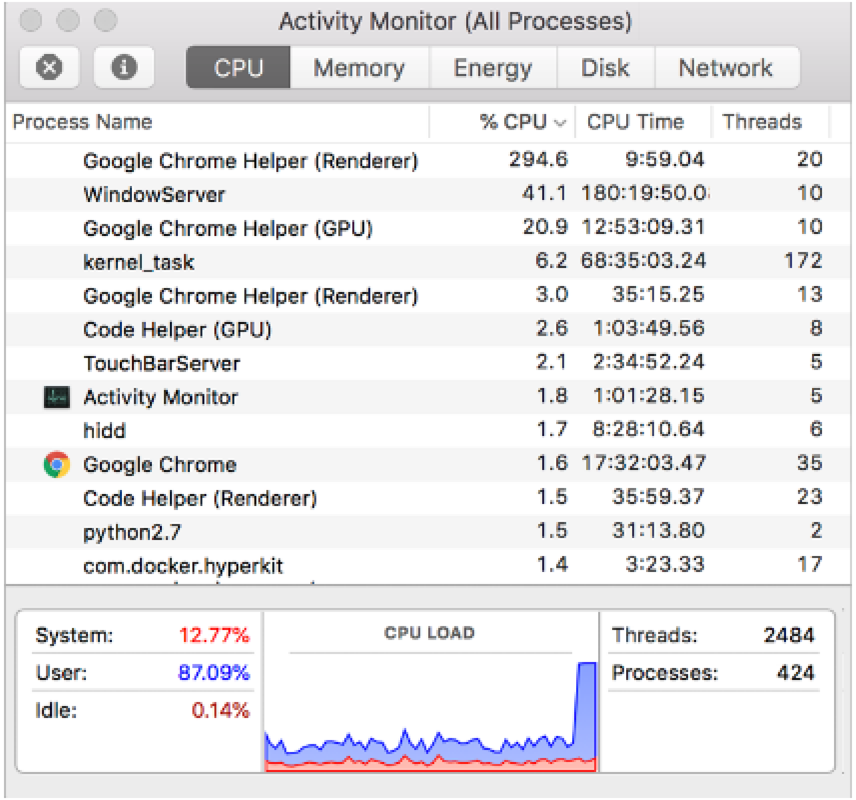
- we can also do parallel computing in our browser.
- Visit the link below to try a parallelized cat detection program:
- https://pamelafox.github.io/parallel-demo/

- You can watch the workers progress in the chart on the webpage.
- The program starts off with a short setup, a sequential portion of initializing the images array and queuing up the tasks.
- Then the workers are off to the races!

Another example

- On many computers, you can also monitor your CPU activity at the same time so that you can see how your CPU is being utilized and how the work is spread across the cores of your CPU.





Another example

- First, we run the program with the maximum number of images for each number of workers and record the duration each time

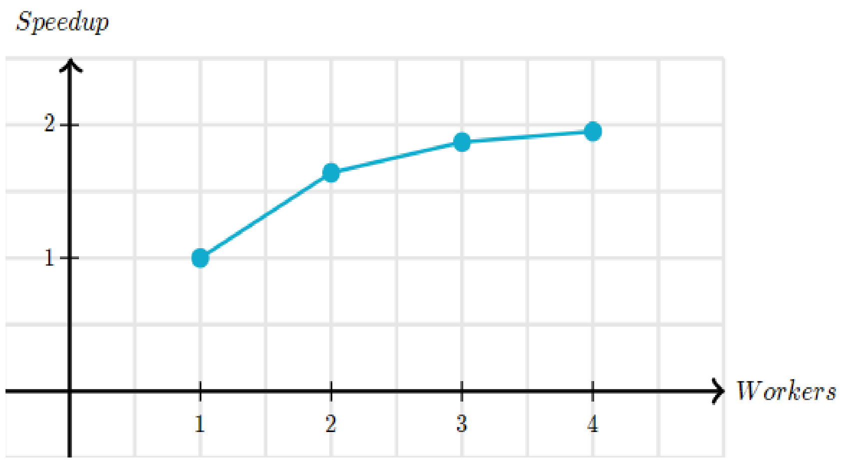
Workers	Duration (seconds)
1	53.91
2	32.95
3	28.81
4	27.66

- Running the program sequentially is basically the same as running the program with a single worker, so we can calculate the speedup by dividing the first duration by each of the other durations.

Workers	Duration (seconds)	Speedup
1	53.91	1
2	32.95	$(53.91/32.95) = 1.64$
3	28.81	$(53.91/28.81) = 1.87$
4	27.66	$(53.91/27.66) = 1.95$

Another example

- We can also graph the speedup to visualize how it changes as the number of workers increases
- The computer got close to a 2x speedup but nowhere near a 4x speedup, which is what we might have expected with 4 workers. Why not?

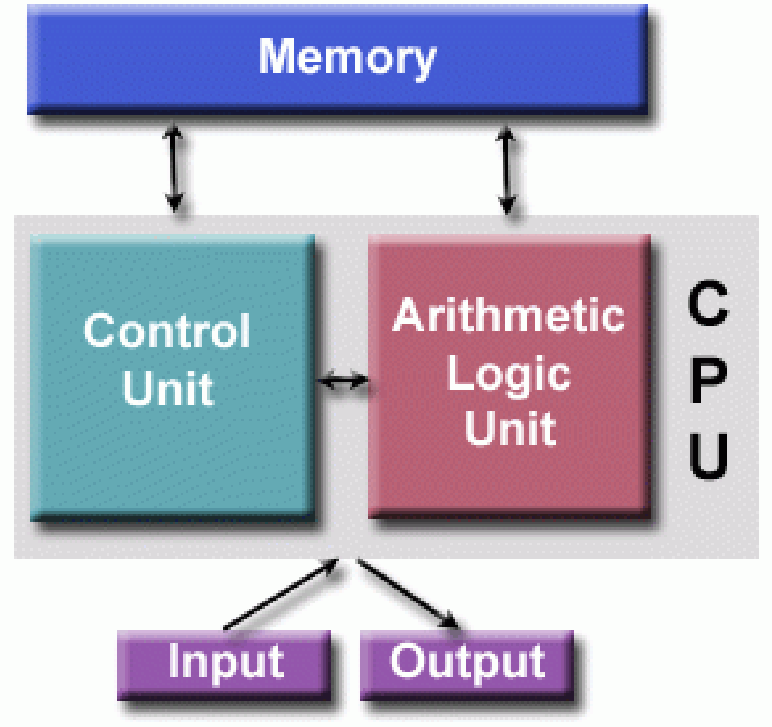


Another example

- Factors that affect performance
  - **Hyperthreading:** Even though my computer reports that it can run four threads concurrently, I discovered that my CPU only has *two* cores
  - **Other CPU activity:** When this program runs from a web browser on a computer, it's competing for CPU time with other processes.
  - **User interface updates:**
    - The webpage that runs this program includes many visual elements: the constantly updating chart, the images and their loading indicators, the status text.
    - Whenever a webpage needs to update a visual element, the CPU is doing work to calculate the new pixels and render them to the screen.
    - That additional work slows down the execution time.

Different parallel computers

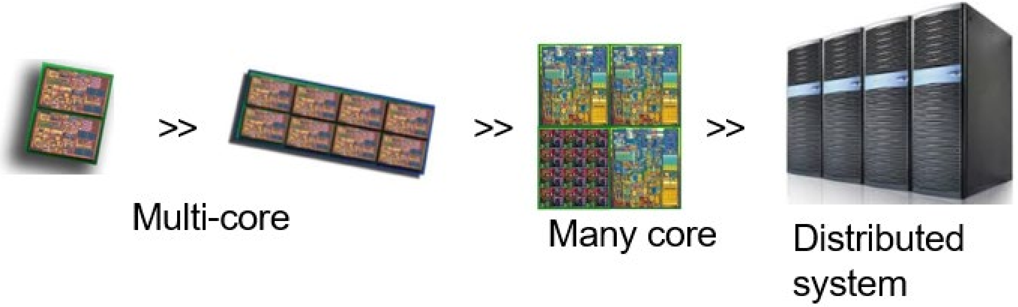
- Named after the Hungarian **mathematician John von Neumann** who first authored the general requirements for an electronic computer in his 1945 papers.



Different parallel computers

- Since different parallel computers have different characteristic, there is need for design and development of new and efficient parallel algorithms and programs for different parallel computers.

Parallelism



Heterogeneous coprocessors

Different software and platform

AM++	Copperhead	ISPC	OpenACC	Scala
ArBB	CUDA	Java	PAMI	SIAL
BSP	DryadOpt	Liszt	Parallel Haskell	STAPL
C++11	Erlang	MapReduce	ParalleX	STM
C++AMP	Fortress	MATE-CG	PATUS	SWARM
Charm++	GA	MCAPI	PLINQ	TBB
Chapel	GO	MPI	PPL	UPC
Cilk++	Gossamer	NESL	Pthreads	Win32
CnC	GPars	OoOJava	PXIF	threads
coArray Fortran	GRAMPS	OpenMP	PyPar	X10
Codelets	Hadoop	OpenCL	Plan42	XMT
	HMMP	OpenSHMEM	RCCE	ZPL

Conclusion

- What is parallel computing
- Why parallel computing
  - High performance
  - Technology push
  - Application driven
- Why study parallel computing