Chapter 1. Introduction to Modern Architectures, Hardware, & Platforms

Contents of this Chapter

- Overview of Computer generations hierarchy
- Embedded Systems – Issues & Implementation options for computing systems
- Elements of Modern Computer Systems
- Classification of Compute Platforms
- Performance Issues
- Programming for Parallelism
- Multiprocessors and Multicomputers
- Multithreading vs Multiprocessing
- Vector & SIMD Supercomputers
- Introduction to PRAM and VLSI models
- Loosely Coupled Architectures – Handling Big Data

References *:*

1) Kai Hwang's Book on Advanced Computer Architecture - Chapter 1 pages 3-46;

*2) Andina et. al., "*FPGAs Fundamentals, advanced features, and applications in industrial electronics", CRC Press, 2017 Edition

3) Kai Hwang et.al," Distributed and Cloud Computing: From Parallel Processing to the Internet of Things" ISBN-13: 978-0123858801 / ISBN-10: 0123858801, Publisher - Morgan Kaufmann book on Cloud Computing

Data sources & sizes – Examples of some real-life applications

- Social media data - 500++ terabytes everyday added to the DB (Ex: FB, Twitter, etc)
- Jet Engine – ~ every 30 mins generates, 10-12 terabyes of data per engine! Total data then scales easily into Peta/Exa-scales!!
- Weather forecast – on a conservative scale, 1.5 terabytes of meteorological data collected each day by certain countries specific to certain areas;

Computational demand – Some examples

(A) Weather prediction models: Mostly via partial differential equations!

- 180 Latitudes + 360 Longitudes
- From the earth's surface up to 12 layers (h_1,..., h_{12})
- 5 variables – air velocity, temperature, wind pressure, humidity, & time

What does this computation involve? Solving PDEs recursively some **N** *# of times!!*

# of Floating point operations = # of grid points x # values per grid point x # of trials reqd., x # operations per trial

$= (180 \times 360 \times 12) \times (5) \times (500) \times (400) = 7.776 \times 10^{11}$ (!!)

Say, each FP takes about 100 nsecs, then the total simulation time – 21.7 hrs!!! With 10nsecs, simulation time - 2.17hrs!; With 5 nsecs, time = 0.217 hrs ~= 780 secs ~=13 mins!

(B) Data Mining:

- Tera bytes of data (10^{12} bytes)

Complexity of "processing" – Size of the database
x # of instructions executed to check a rule
x Number of rules to be checked
= 10^{12} x 100 x 10 *(very modest!)* = *10^{15} !!*

(C) Satellite Image processing

Edge detection - Pixel by pixel processing –
size of the image x mask size x # of trials
(3000 x 3000) x (11 x 11) x 100 ~ 10^{11} problem size

Both the above problems assume that we are processing with zero communications delays!!

Intel CPU Microarchitectures – Generations hierarchy

Year	Micro-Arch	Pipeline stages	Clk Freq	Node
2021	*Gracemont*	20 unified with misprediction penalty	4000	Intel 7
2021	Golden Cove	12 unified	5500	Intel 7
2021	Cypress Cove	14 unified	5300	14 nm
2021	Alder Lake			10 nm
2020	Willow Cove	14 unified	5300	10 nm
2020	*Tremont*	20 unified	3300	10 nm
2020	Tiger Lake			10 nm
2019	Sunny Cove	14–20 (misprediction)	4100	10 nm
2019	Ice Lake			10 nm
2019	Comet Lake			14 nm
2018	Cannon Lake	14 (16 with fetch/retire)	3200	10 nm
2017	*Goldmont Plus*	? 20 unified with branch prediction ?	2800	14 nm
2017	Coffee Lake			14 nm

2022 – Raptor Cove;
12 Unified;
6GHz /Intel 7

2023/2024 – 3nm tech underway

Ref: https://en.wikipedia.org/wiki/List_of_Intel_CPU_microarchitectures

Moore's Law

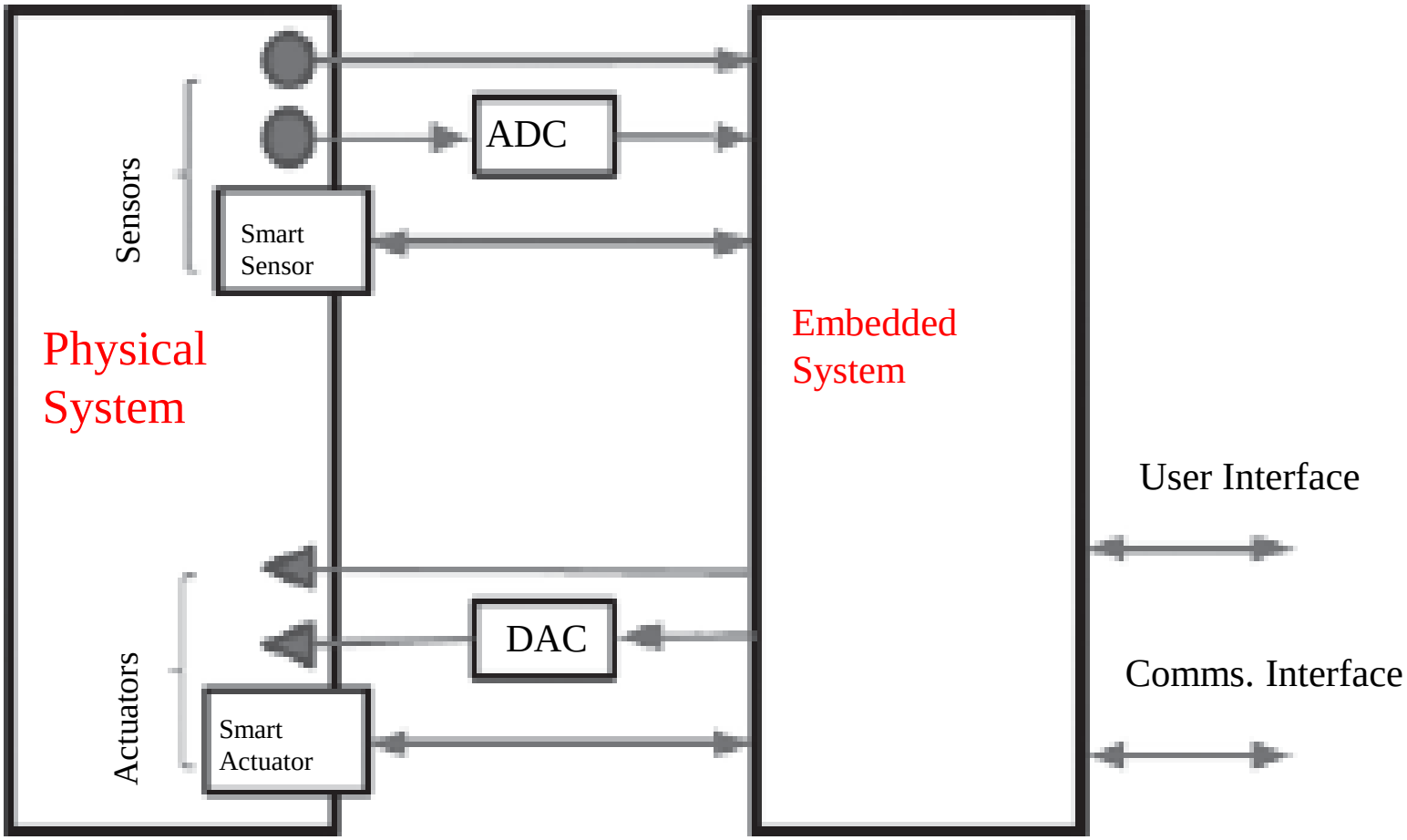
Moore's Law is a computing term which originated around 1970; the simplified version of ***this law states that processor speeds, or overall processing power for computers will double every two years***.

This seems to be very much applicable for transistor growth than the processor speeds (*We will see the trend and reasons in Chapter 3*).

Comparing Computing Power - Your mobile vs. Supercomputer!

- A single Apple iPhone 5 has 2.7 times the processing power than the 1985 Cray-2 supercomputer;
- Early Nintendo Entertainment System (NES) had half of the processing power as the computer that brought Apollo to the moon!

Embedded Systems



Issues & Implementation options for Embedded Systems

Major factors to consider:

- Energy Issues
- Performance vs Flexibility
- Design techniques & tools for different hardware technologies
  - Microcontrollers/CPUs
  - DSP Processors
  - Multicore Processors & GPGPUs
  - FPGAs
  - ASICs
- Technological Improvements and Complexity Growth

Current day concerns – Energy

Embedded Computing, Energy Aware Computing (*Going Green!)*

Walking with Computing Power!! - Phones, game CPUs small scale devices have computing abilities!! [CPU + Memory capacity + Thin codes] ☺

Energy optimizations – key requirement for modern day computing machines and Storage centers!!

Hardware ASICs – at least 50X more efficient than programmable processors; The latter uses more energy (instructions and data supply) – Earlier designs (Stanford ELM Processor) attempts to minimize these energy consumptions!

Key idea – Supply instructions from a small set of distributed instruction registers rather than from cache! For data supply, ELM places a small, four-entry operand register file on each ALU's input and uses an indexed file register!!

Microarchitecture design offers more challenges for more performance optimization.

Energy Issues – Hardware and Software Engineering

- Hardware optimizations – Key concern in low-power consumption via:
  - *supply voltage reduction, transistor sizing, operating frequency control*

Processor Clock frequency f can be expressed in terms of supply Voltage V_{dd} a threshold voltage V_T as follows:

$$f = k\,(V_{dd} - V_T)^2 / V_{dd} \qquad (1)$$

From (1) we can derive V_{dd} as a function of f as $F(f)$:

$$F(f) = (V_T + f/2k) + \textbf{sqrt}\,((V_T + f/2k)^2 - V_T^2)) \qquad (2)$$

The processor power can be expressed in terms of frequency f , switched capacitance C and V_{dd} as follows:

$P = 0.5 .f.C.V^2_{dd} = 0.5 . F. C. F(f)^2$ (3)

(3) can be proven to be a convex function of *f*. Thus we see a Quadratic dependence between P & V_{dd} . This leads to the design of Dynamic Voltage scaling processors wherein there exist a set of voltage values that can control processor operating frequencies (speeds). *Consequence?*

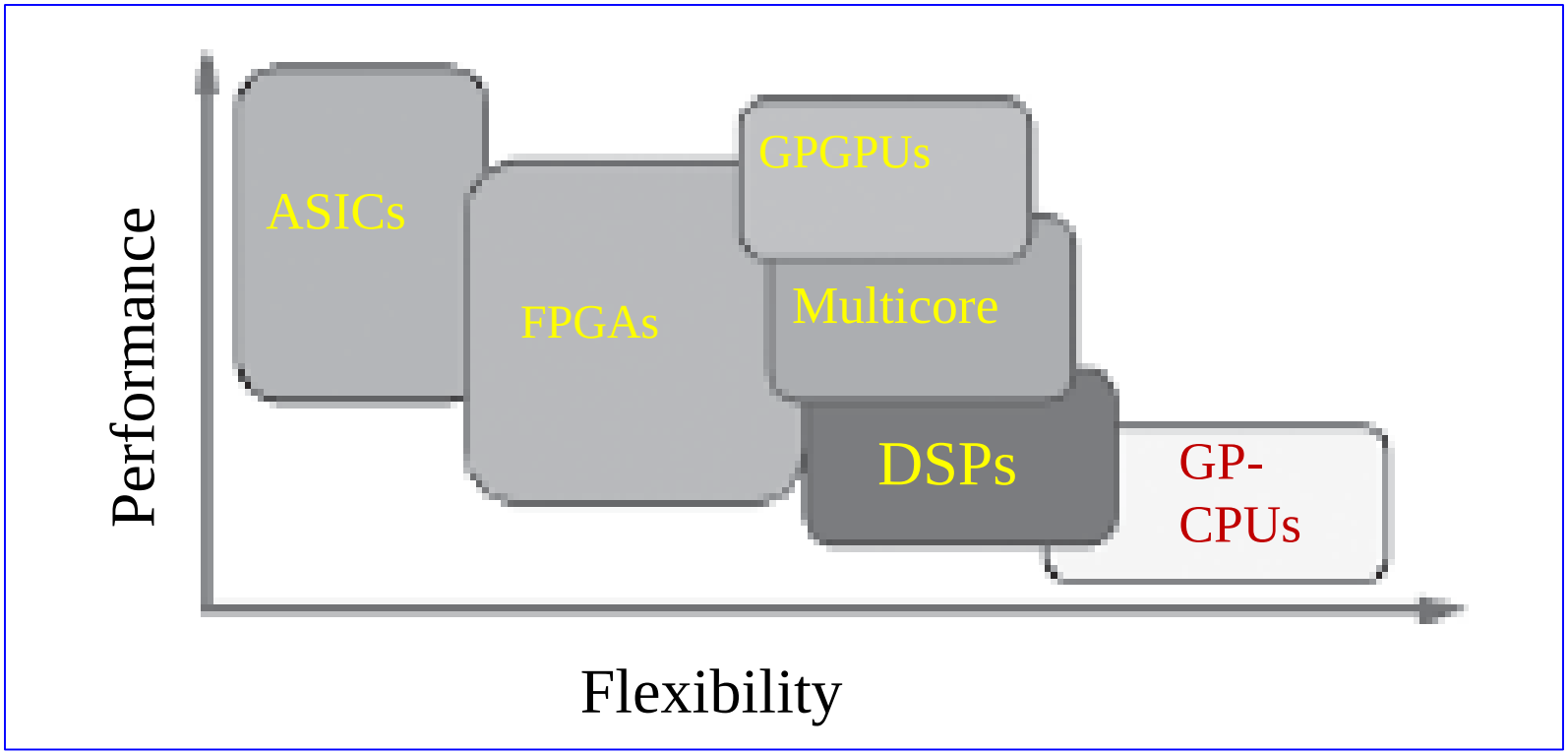
In scheduling computationally intense tasks, a third dimension comes into play *implies* time & power trade-off!!

So algorithms are designed to strike this trade-off thus meeting the respective budgets (time and power).

Energy Issues – Hardware and Software Engineering

- Software design emphasis for energy reduction
  - Software decisions are usually addressed at higher levels of the design hierarchy and therefore the resulting energy savings are larger;
  - Software energy optimization aims at implementing a given task using fewer instructions and in this way there is almost no trade-off between performance and power as in the case of hardware
  - Other s/w methods target memory related power consumption and apply code transformations to reduce the number of accesses to memory layers

- Performance vs Flexibility – *Different hardware platforms*



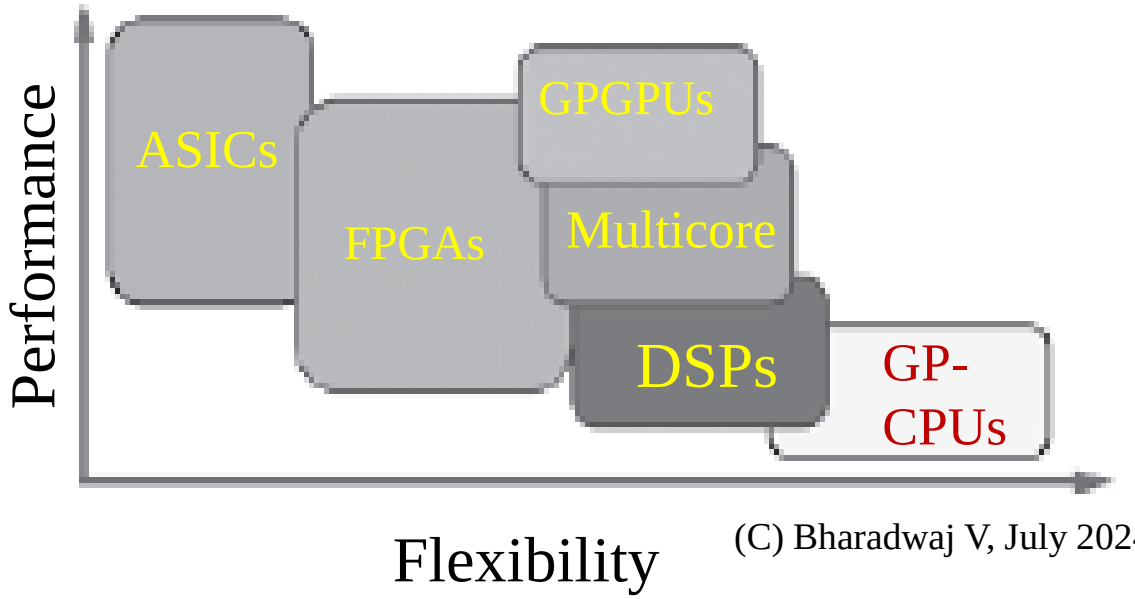
What does *flexibility* mean?

- Flexibility is somewhat related to the ease of use of a system and its possibilities to adapt to the changes in specifications.

Some points to observe from the qualitative depiction to choose a compute system

1) Highest performance - ASICs & GPGPUs

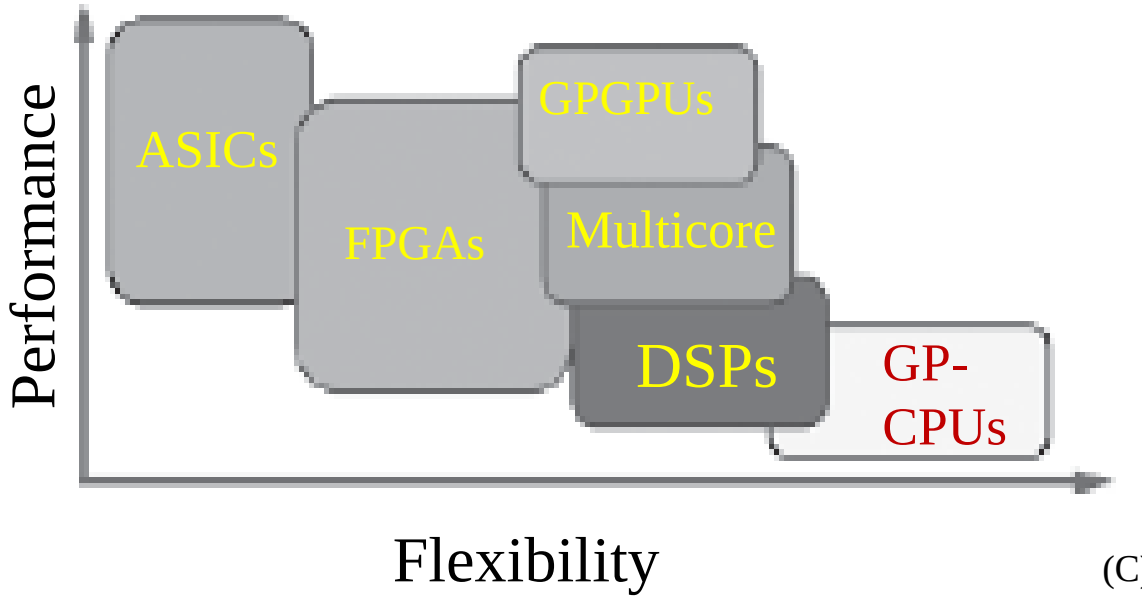
2) Lack of flexibility of ASICs is compensated by their high energy efficiency. In contrast, GPGPUs are excellent in performance (assisted with software too) but extremely power consuming!



Some points to observe from the qualitative depiction to choose a compute system

3) Multicore architectures are somewhat closer to GPGPUs' performance especially when their inherent parallelism matches the target application

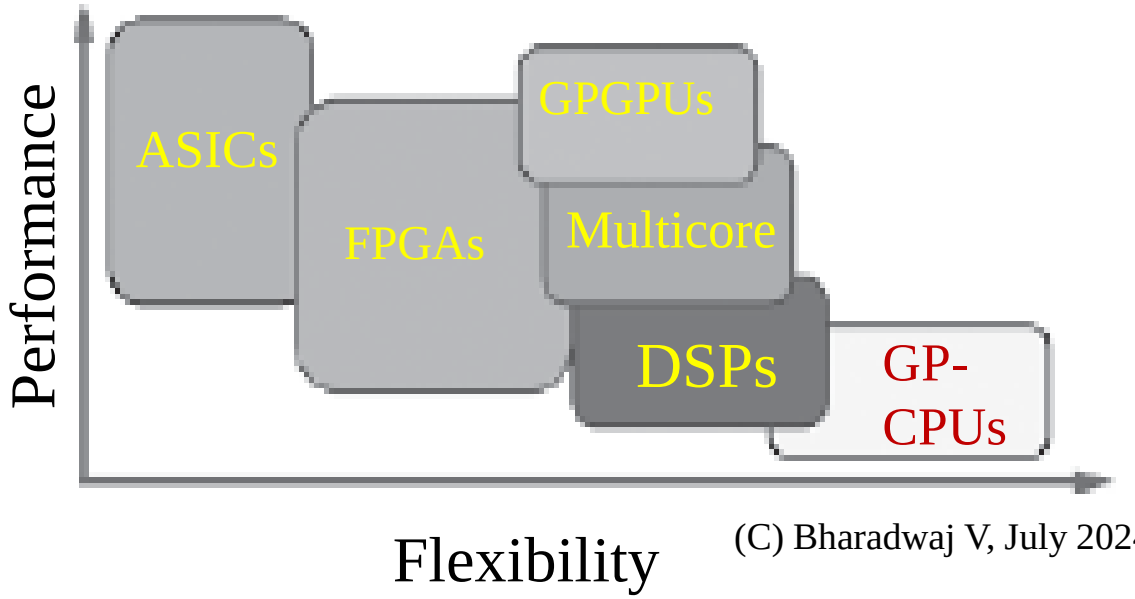
4) But GPGPUs thoroughly exploit data parallelism whereas multicore systems are best suited for multi-tasking



Some points to observe …. (Cont'd)

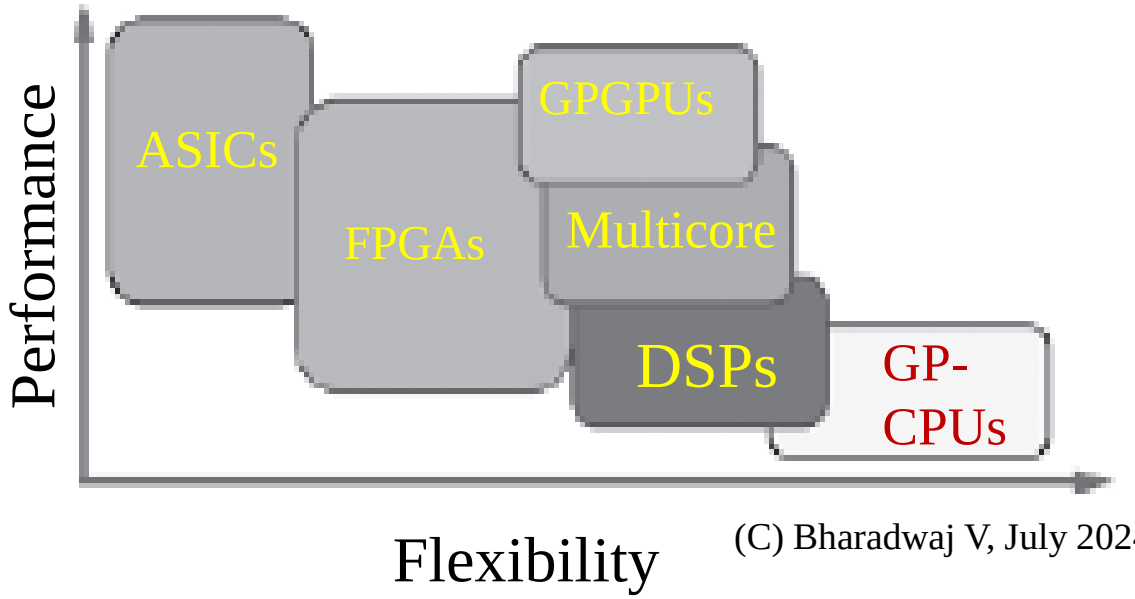
5) On flexibility, DSPs are much better than single general purpose micro-processors/controllers because DSPs are custom-designed to handle specific functions

6) Compared to single general purpose micro-processors/controllers FPGAs are less flexible because the former ones adopt software-based solutions than hardware-based solutions that exist in FPGAs.

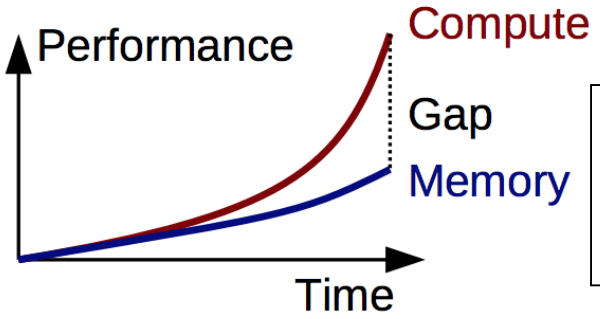


Some points to observe …. (Cont'd)

7) However, there exist FPGAs that can be reconfigured during operation, which may ideally be regarded to be as flexible as software-based platforms since reconfiguring an FPGA essentially means writing new values in its configuration memory. So, FPGAs are considered in between ASICs and software-based systems, in the sense that they have hardware-equivalent speeds with software-equivalent flexibility.



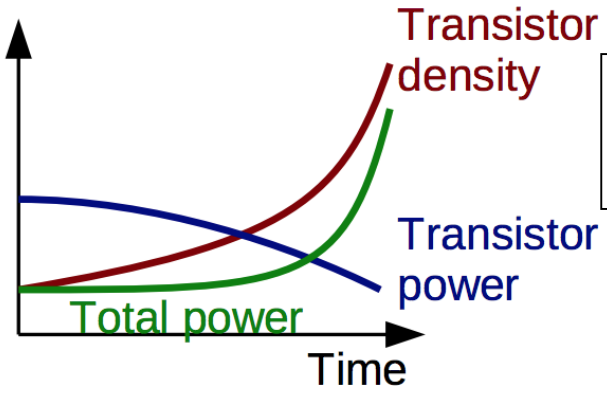
Performance vs Flexibility



Memory performance

Memory speed does not increase as fast as computing speed; More and more difficult to hide memory latency; Gap is smaller in using new NVMe SSDs;

Note: The graphs are shown for General purpose CPUs

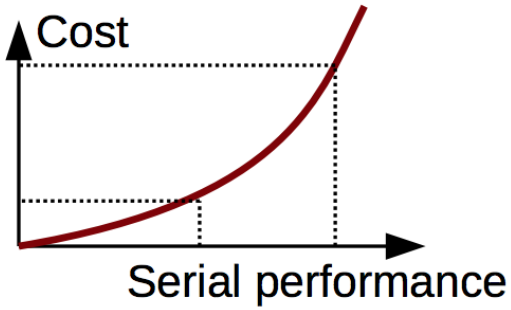


Power performance

Power consumption of transistors does not decrease as fast as density increases; Performance is now limited by power consumption

Thus, to maximize the performance we need a fusion of several concepts.

Consequently…



Instruction Level Parallelism Performance

Pollack's rule - Performance increase due to microarchitecture advances is roughly proportional to the square root of the increase in complexity

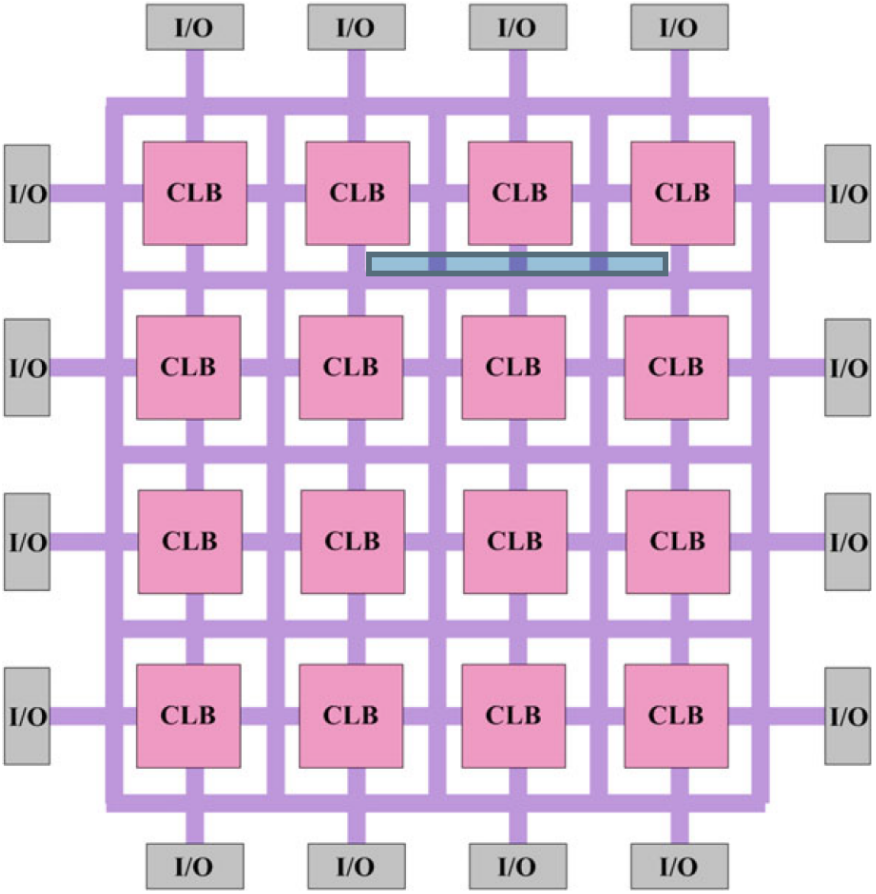
Two approaches

(A) Flexible hardware-cum-programming

(B) Hardware acceleration

(A) Flexible hardware-cum-programming

An example platform: FPGA Architecture

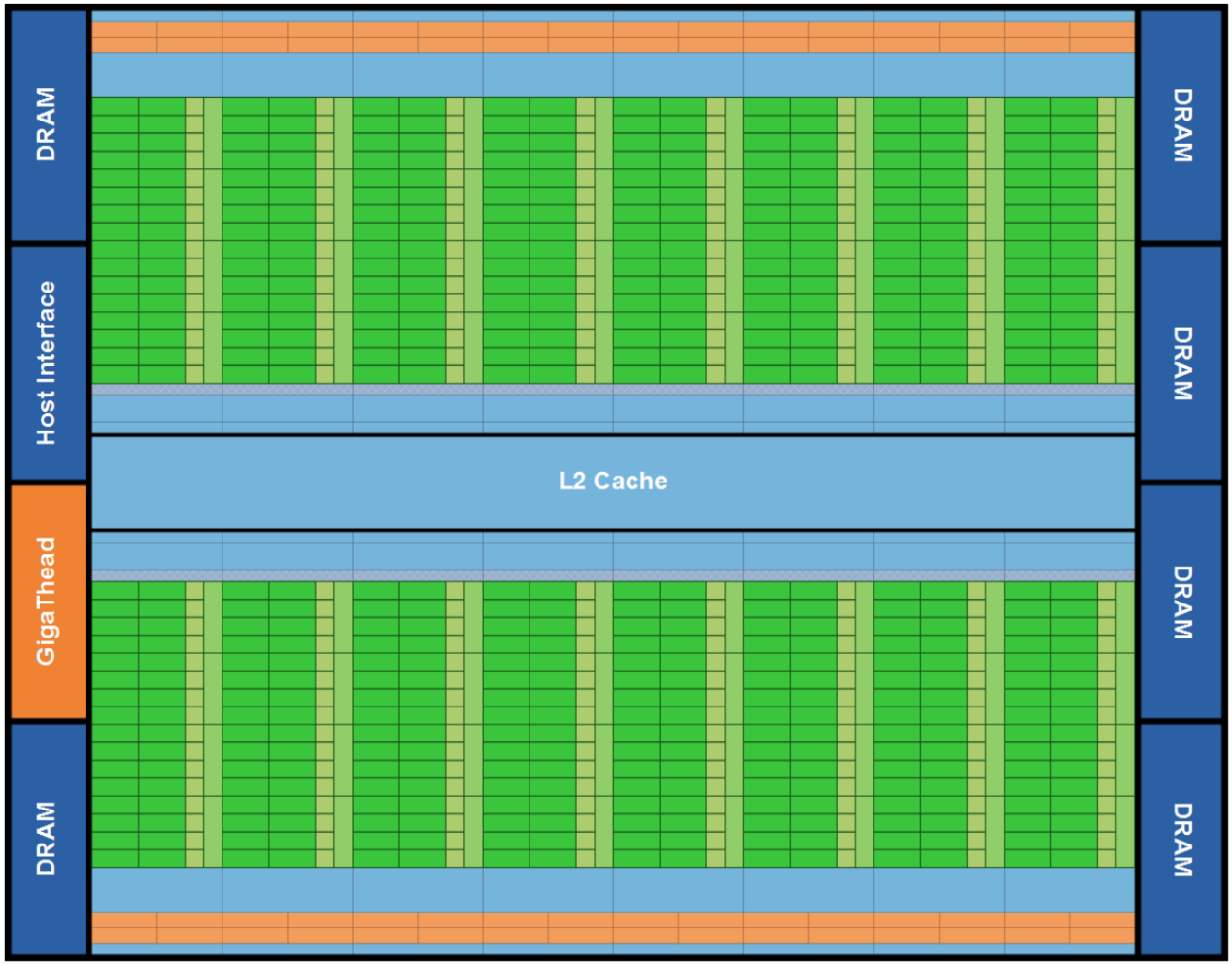


Normally FPGAs comprise of:

- Programmable logic blocks which implement logic functions.
- Programmable routing that connects these logic functions.
- I/O blocks that are connected to logic blocks through routing interconnect and that make off-chip connections.

Quick Primer - https://www.ni.com/en-sg/shop/electronic-test-instrumentation/add-ons-for-electronic-test-and-instrumentation/what-is-labview-fpga-module/fpga-fundamentals.html

Example (Classical) GPU - FERMI CUDA Architecture



Even earlier versions – 3.0 billion transistors, features up to 512 CUDA cores

Earlier CUDA core - Executes a floating point / integer instruction per clock for a thread. The 512 CUDA cores are organized in 16 SMs of 32 cores each

*(**Chapter 5**! Stay tuned!)*

With CUDA, researchers and software developers can send C, C++, and Fortran code directly to the GPU without using assembly code. This allows them take advantage of parallel computing in which thousands of tasks, or threads, are executed simultaneously.

(B) Hardware Acceleration (HA)

- HA means tasks being offloaded to devices and hardware which specialize in it;

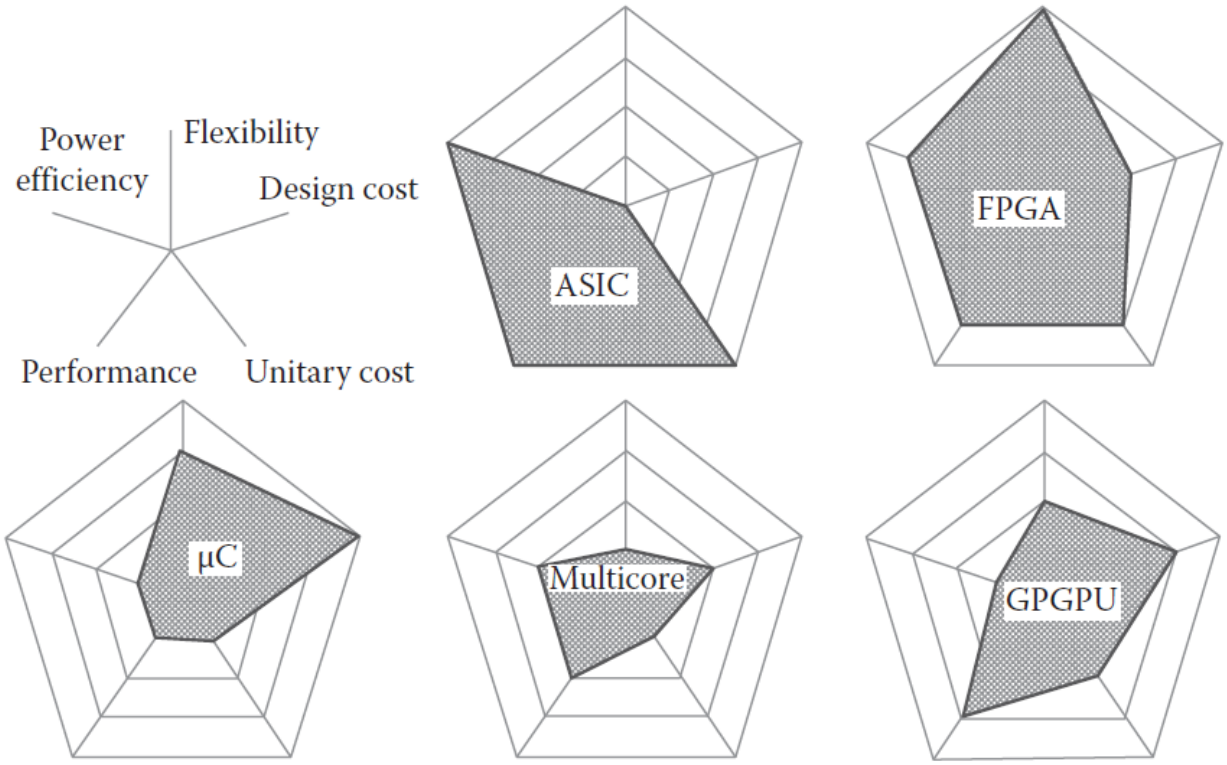
- Modern day CPUs are powerful and hence they are targeted first to run the tasks; However, for efficiency and performance dedicated hardware is used (Ex., sound cards, graphics units, etc)

Few points to consider…

If your CPU is powerful and task running is not compute intensive, use of HA remains insignificant or no use; For graphics display and interactive ambience (games) enabling HA can render an awesome experience!

- **Design techniques & tools for different technologies**
  - Microcontrollers/CPUs
  - DSP Processors
  - Multicore Processors & GPGPUs
  - FPGAs
  - ASICs

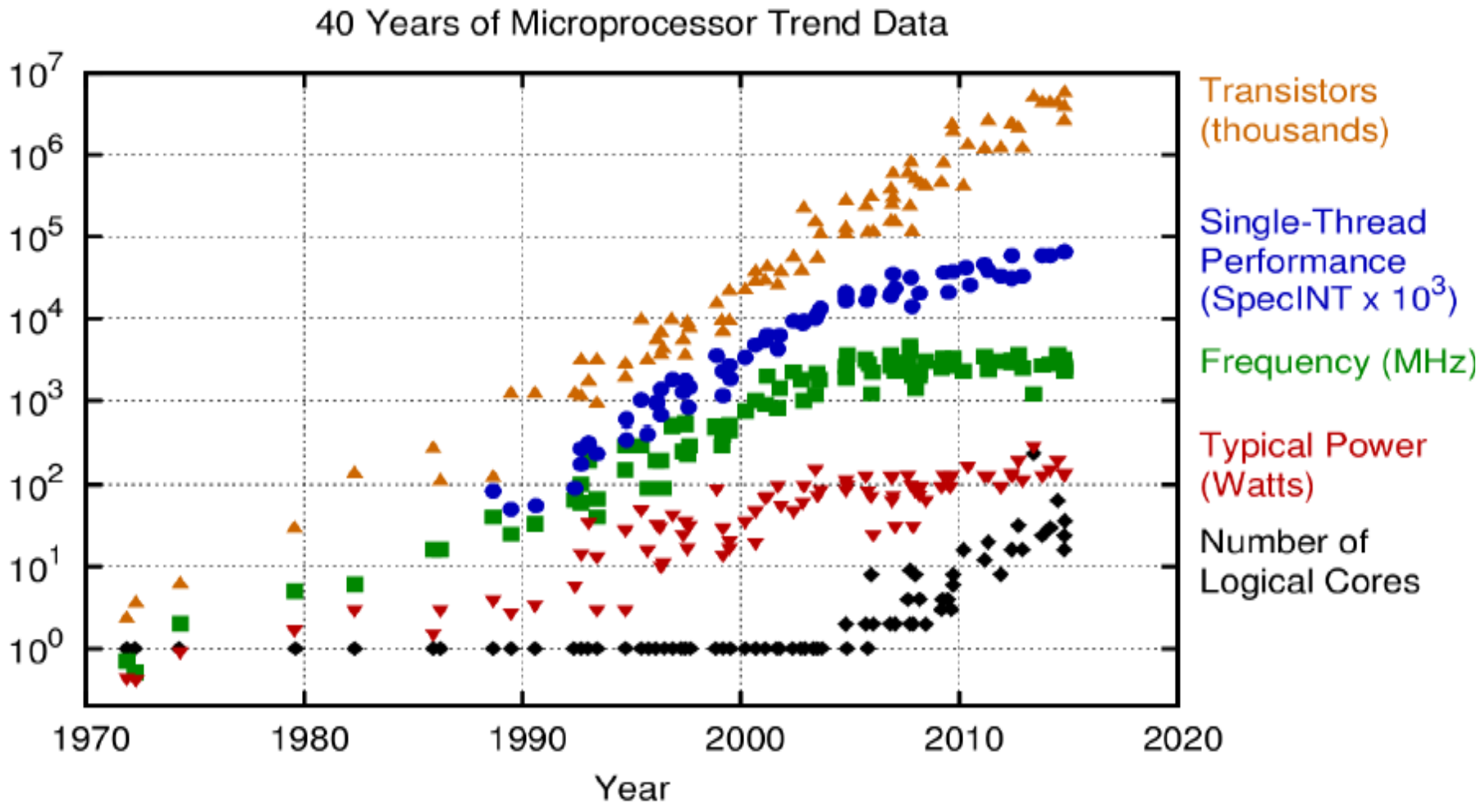
Comparative features - Outer (further away from the center) is better for performance related parameters (other than cost)



- Technological Improvements/Trends and Complexity Growth

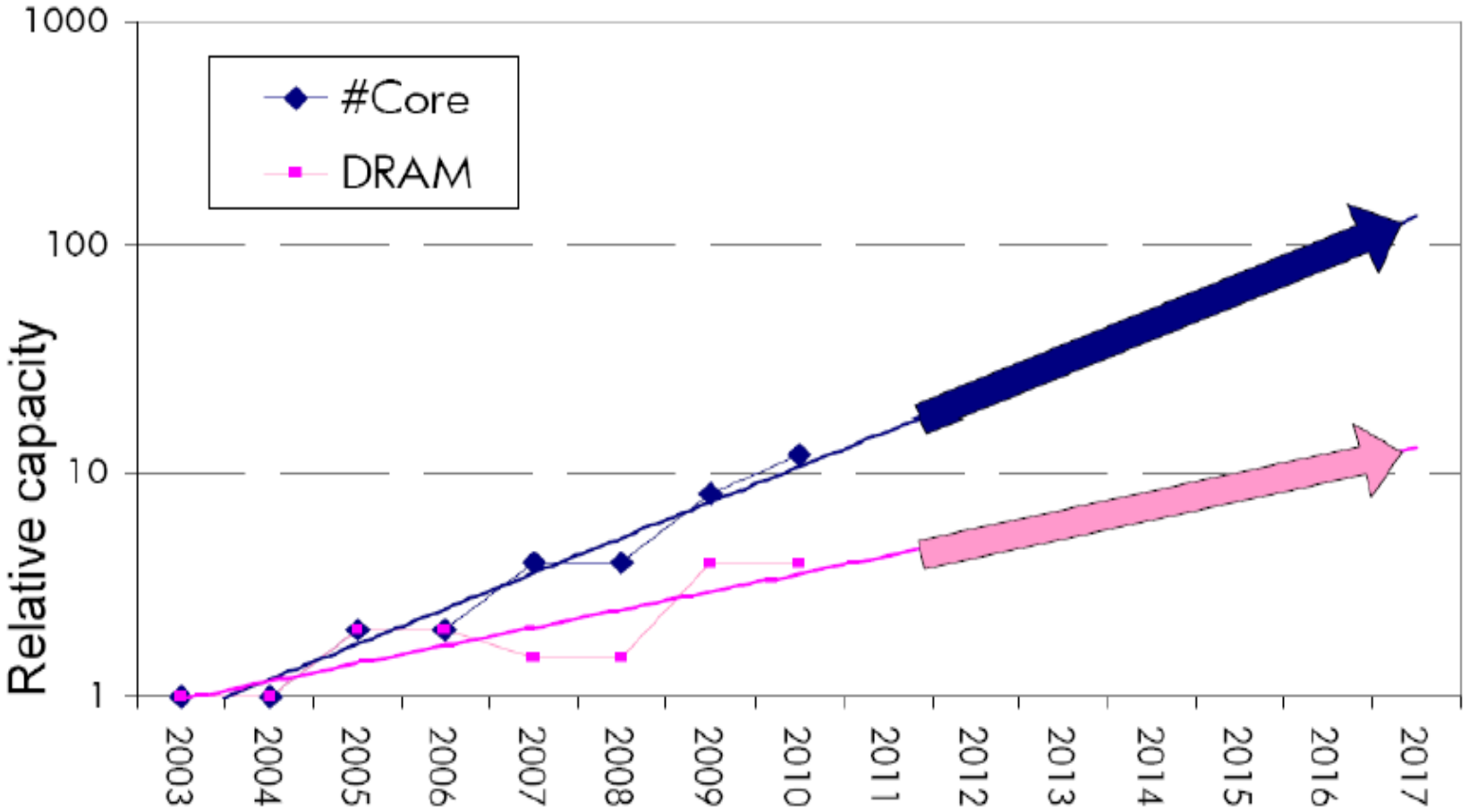
- Technology trend on current day compute platforms
- Trend on Memory capacity & bandwidth – Specifically on tech growth in infusing number of cores and memory capacity demands
- Network technology – supporting technology
- Energy & Security

Trend Data on current day compute platforms



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

Trend Data on Memory Capacity & Bandwidth



Network technology – Performance depends on Switches + transmission system; It took about 10years for Ethernet technology to mature to move from 10Mb to 100Mb; 1Gb was made available after 5 years; multiprocessor networks – loosely coupled systems have become common these days.

Although costs of integrated circuits have dropped exponentially, the basic procedure of silicon manufacture is unchanged. A *wafer* is still tested and chopped into *dies* that are packaged. Thus the cost of a packaged IC is:

Cost of a IC
= cost of (die + testing a die + packaging and final test) / Y

where Y is the final test yield

Current day concerns – Secured hardware systems

Hardware vulnerability - Exploitable weakness in a computer system for *attacking*; A hardware is said to be *vulnerable*, if by any means by which "a code" can be introduced to a computer!

Attack is enabled via remote or physical access to system hardware

Different types by which a hardware can be fiddled!

- Attack via copying malicious files to your memory via flash drives, CD, etc.
- Due to an unexpected flaw in operation that allows attackers to gain control of a system *by elevating privileges or executing code*; These vulnerabilities can sometimes be exploited remotely, rather than requiring physical access.

Hardware vulnerability …. (Cont'd)

Example of such an exploit is "*Rowhammer*":

- *How does it work?* This works by repeatedly rewriting memory in the same addresses to allow retrieval of data from nearby address memory cells – even if the cells are protected!

- Ideally this is not supposed to happen, it can and it does due to hardware flaws that are hard to prevent. *Proof-of-Concepts* developed by researchers demonstrate that by repeatedly accessing a row of memory can cause bit flips in adjacent rows of some DRAM devices. This means, "spatial access" property is used to steal information from adjacent rows!

Hardware vulnerability …. (Cont'd)

In general, hardware vulnerabilities are not exploited through random hacking attempts. They are targeted attacks of known high-value systems and organizations.

For common users, traditional malware protection and a locked door are sufficient protection! ☺

Top Supercomputers (June 2024 listing)

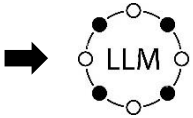
Check this out -> **https://www.top500.org/lists/**

Above link has three categories:

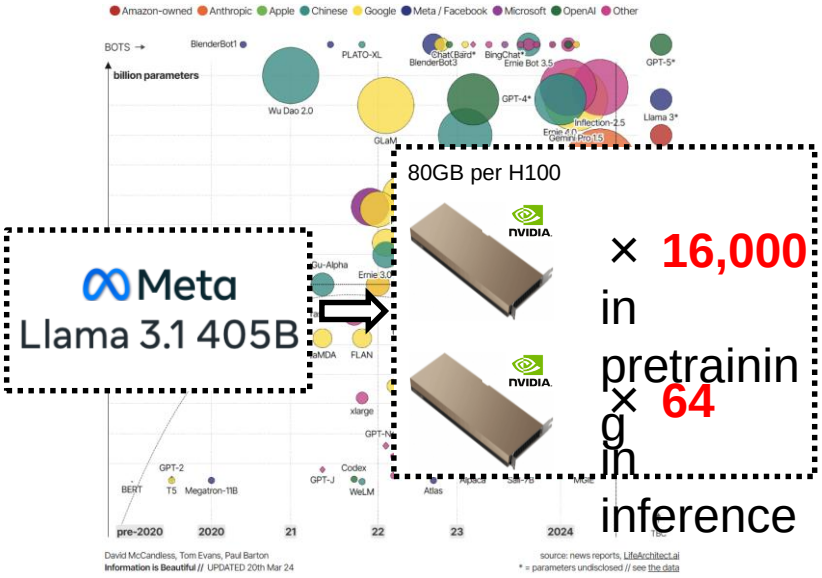
- Top 500 – Computing related aspects
- HPC – Related to HPC computing
- Green500 – Related to energy optimized performance

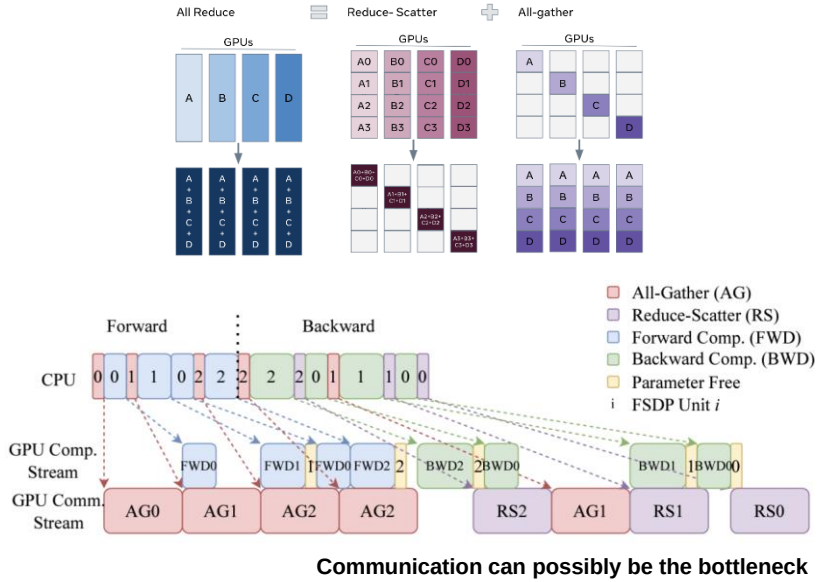
Demands in Advanced Algorithms – LLM as an Example

[In] What are the three primary colors?



[Out] The three primary colors are red, blue, and yellow.





Communication can possibly be the bottleneck

❶ Memory capacity bound

❷ Communication bound

Courtesy: Yimin Wang (PhD Scholar ECE Dept)

Demands in Advanced Algorithms – LLM as an Example

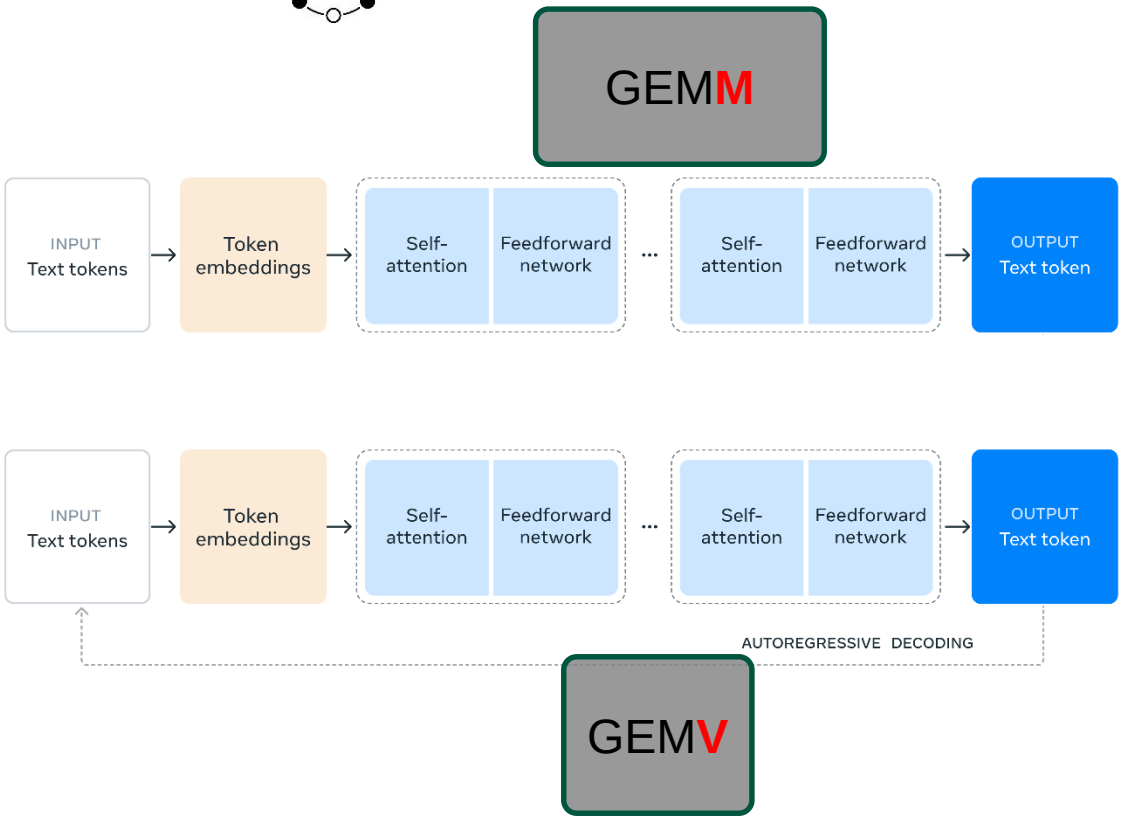
[In] What are the three primary colors? → LLM → [Out] The three primary colors are red, blue, and yellow.

Phase 1: Prompt processing

❸ Computation bound

Phase 2: Token generation

❹ Memory bandwidth



Courtesy: Yimin Wang (PhD Scholar ECE Dept)

We are now in the era of Quantum Computing! Very soon this module will expose Quantum Computing Architecture, as the design and performance characterization are still in infancy!

Elements of Modern Computer Systems

Problems:

1. Need of different computing resources

2. Numerical computations - CPU intensive

3. AI type of problems, transaction processing, weather data, sesimographic data, oceanographic data, etc

Fundamental elements in a compute platform

- Algorithms and Data Structures
- Hardware
- OS
- System Software Support

[HLL - compiler (object code) - assembler - loader]

- Efficient Compilers - pre-processor, pre-compiler, parallelizing compilers

Compiler upgrading approaches:

Preprocessor - Uses a sequential compiler and a low-level library of the target computer to implement high-level parallel constructs

Precompiler (parallel) - This requires some program flow analysis, dependency checking, detection of parallelism to a limited extent.

Parallelizing compilers - This demands the use of a fully developed parallel compilers that can automatically detect the embedded parallelism in the code.

Compiler directives are often used in the source code to help the compilers for efficient object codes and speed.

- **Evolution of computer Architecture –** *Chap1 Scanned slides*

Flynn's Classification(1972)

1. SISD (Single Instruction Single Data) - Conventional sequential machines

2. SIMD (Single Instruction Multiple Data) - vector computers example - connection machine with 64K 1-bit processors; *lock-step* functional style

3. MIMD (Multiple Instruction and Multiple Data) - parallel computers are reserved for this category

4. MISD (Multiple Instruction Single Data) - systolic arrays

Above classification is only based on the way an architecture functions!

Parallel /Vector Computers (MIMD mode of operation)

Two types: (a). Shared-memory multiprocessors
(b). Message-Passing multi-computers

Multiprocessor/Multi-computer features are as follows.

MPS: communicate through shared variables in a common memory
MCS: no shared memory; message passing for communication

Vector Processors - Equipped with multiple vector pipelines; Control through hardware and software possible; These are of two types:

(a). Memory to Memory architecture[mem to pipe and then to mem]
(b). Register to Register architecture [mem - vector regs - pipe]

For synchronized vector processing, we use SIMD computers.

Predominantly SIMD exploits **spatial parallelism** rather than **temporal parallelism** (example: pipeline processing). SIMD uses an array of PEs synchronized by the same controller.

Computer System Development: *Refer to the scanned pages pdf*

Performance Issues

The best/ideal performance of a computer system demands a perfect match between machine capability and program behavior.

Best programming + data structures + efficient compilers + minimizing the overhead processing(disk access, etc)

Program performance includes the following cycle, typically.

Disk/Memory accesses, input/output activities, compilation time, OS response time, and the actual CPU time.

In a sequential processing system, one may attempt to minimize these individual times to minimize the total turnaround time, as we have no choice! $T^* = \min \Sigma T_k$, k=1,2,..,n (n: # of events)

In a multi-programmed environment, I/O and system overheads may be allowed to overlap with CPU times required in the other programs. p: # of CPUs/processes/programs….

$T^* = \min \max \{T_k\}$ where max is over k=1,2,..,n, and we minimize that T_p

So, a fair estimate is obtained by measuring the exact number of CPU clock cycles, or equivalently, the amount of CPU time needed to compute/execute a given program.

(a). Clock rate and Cycles per instruction (CPI):

Clock cycle time : τ expressed in nanoseconds
Clock rate: ***f*** expressed in megahertz or GHz (inverse of τ)

The size of the program is determined by its *instruction count* **(I_c) --** in terms of number of machine instructions (MIs) to be executed in the program.

Different MIs may require different number of clock cycles, and hence, *clocks per instruction* (CPI) is an important parameter.

Thus, we can easily calculate the average CPI of a given set of instructions in a program, provided we know their frequency. It is a cumbersome task to determine the CPI exactly, and we simply use CPI to mean the "average CPI", hereafter.

(b). CPU time:

The CPU time needed to execute the program is estimated by

$T = I_c \cdot CPI \cdot \tau$ (# of instr.* clks per instr. * clk cycle time) **(1)**

The execution involves: inst. fetch -> decode -> operand(s) fetch -> execution --> storage

Memory cycle - Time needed to complete one memory reference. By and large, the memory cycle $= k\,\tau$, and the value of k depends on the memory technology. We can rewrite (1) as,

$$T = I_c \cdot (p+mk)\,\tau \qquad (2)$$

where,

p : number of processor cycles needed for instruction decode and execution

m : number of memory references needed

I_c : Instruction count

τ : processor cycle time

All these parameters are strongly influenced by the system attributes mentioned below

(1). Instruction-set architecture - I_c and p

(2). Compiler technology - I_c, p and m

(3). CPU implementation and control - p. τ (total CPU time needed)

(4). Cache and memory technology - k. τ (total memory time)

(Refer to Table 1.2 - page 16)

(c). MIPS rate

C: number of clock cycles need to execute a program

Using (2), the CPU time is given by, $T = C.\ \tau = C/f$

Further, $CPI = C / I_c$ and $T = I_c.CPI.\ \tau = I_c.CPI / f$

The processor speed is usually measured in terms of million instructions per second (MIPS).

We refer to this as "MIPS rate" of a given processor. Thus,

$$\textbf{MIPS rate} = I_c/(T.\ 10^6) = f / (CPI.\ 10^6) = f.I_c / (C.10^6) \qquad (3)$$

Thus, based on the above equation, we can express T as

$$\mathbf{T = I_c \,.\, 10^{-6}/MIPS}$$

Thus, the MIPS rate of a given computer is *directly proportional to the clock rate and inversely proportional to the CPI*.

Note: All the above-mentioned four factors affects the MIPS rate, which varies from program to program.

(d). Throughput rate:

How many programs a system can execute per unit time(W_s)?

Ws: System throughput

CPU throughput $\mathbf{W_p = 1/T = \mathit{f} / (I_c.CPI) = (MIPS) \,.\, 10^6 / I_c}$ (4)

W_p is referred to as CPU throughput.

The system throughput is expressed in programs/second, denoted as W_s. Note that $W_s < W_p$, due to all additional overheads in a Multi-programmed environment. In the ideal case, $W_s = W_p$.

For mobile platforms!

For current day mobile platforms, all the above metrics would be still valid. In addition, the MIPS performance metric should be considered as "MIPS/mW"

Most mobile platforms would be underclocked! (*Why?*)

Best CPU in this case is also the one that gives **max** MIPS/mW.

Programming for Parallelism

(a) Implicit Parallelism (b). Explicit Parallelism

Implicit Parallelism

- *Source Code (Seq versions)*
- *Parallelizing Compiler*
- *Parallel Object codes*
- *Execution by the runtime*

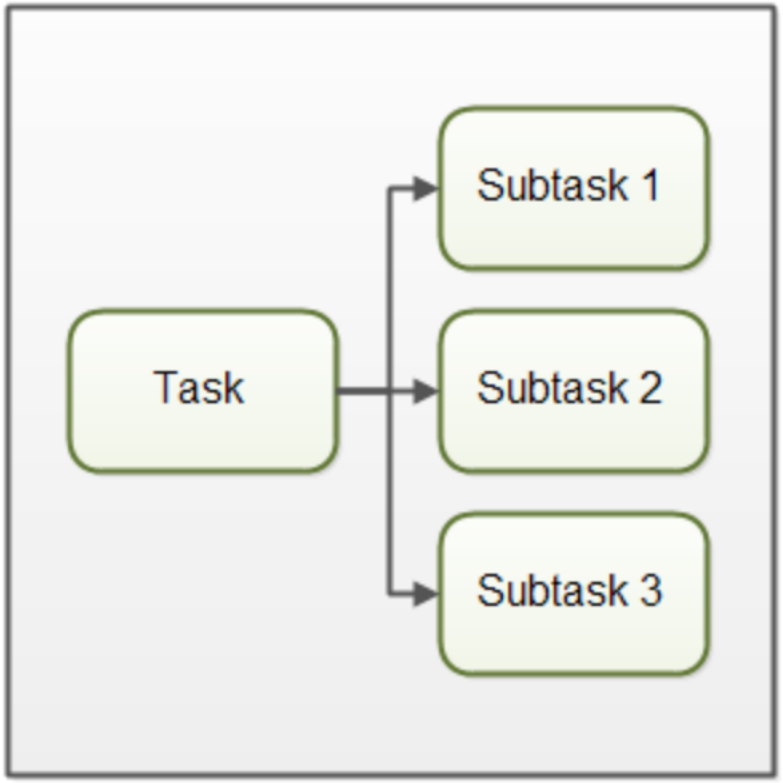
Explicit Parallelism

- *Source Code (Parallel versions)*
- *Concurrency preserving Compiler*
- *Concurrent Object codes*
- *Execution by the runtime*

Refer to Fig. 1.5 on page 18 (Page 6 on scanned slides)

Concurrency vs Parallelism

Concurrency:
Multiple tasks makes progress at the same time.



Parallelism:
Each task is broken into subtasks which can be processed in parallel.

Remarks on Concurrency vs. Parallelism:

Concurrency is related to how an application handles multiple tasks it works on.
An application may process one task at time (sequentially) or work on multiple tasks at the same time (concurrently).

Parallelism on the other hand, is related to how an application handles each individual task.
An application may process the task serially from start to end, or split the task up into subtasks which can be completed in parallel.

- As you can see, an application can be concurrent, but not parallel. This means that it processes more than one task at the same time, but the tasks are not broken down into subtasks.

- An application can also be parallel but not concurrent. This means that the application only works on one task at a time, and this task is broken down into subtasks which can be processed in parallel.

- Additionally, an application can be neither concurrent nor parallel. This means that it works on only one task at a time, and the task is never broken down into subtasks for parallel execution.

Remarks (cont'd):

- Finally, an application can also be both concurrent and parallel, in that it both works on multiple tasks at the same time, and also breaks each task down into subtasks for parallel execution.

However, in this case, some of the benefits of concurrency and parallelism may be lost, as the CPUs in the computer are already kept reasonably busy with either concurrency or parallelism alone. Combining it may lead to only a small performance gain or even performance loss.

So, we need to be very cautious to analyze and measure before we adopt a concurrent parallel model.

Multiprocessors and Multi-computers

1. Shared Memory MPS -- UMA, NUMA, and COMA models

2. Distributed Memory Multi-computers

   Refer to Table 1.4 - different multi-computer systems

Parallel computers - SIMD or MIMD configurations

- SIMD - Special purpose; not scalable
- General purpose computers favor MIMD configuration with distributed memories having a globally shared virtual address space
- Gordon Bell (1992) - Taxonomy for MIMD category

UMA Model: All the processors have equal access times to all the words in the memory, hence the name uniform memory access
Suitable for general purpose and time-critical applications

• *Symmetric processors/system: A*ll the processors have equal access to all the peripherals, and all can execute the system programs(OS) and have I/O capability

• *Asymmetric processors/system*: Only a subset of the processors have the access rights to peripherals and are designated as Master Processors(MPs) and the rest are designated as Attached Processors(APs). The APs work under the supervision of MPs.

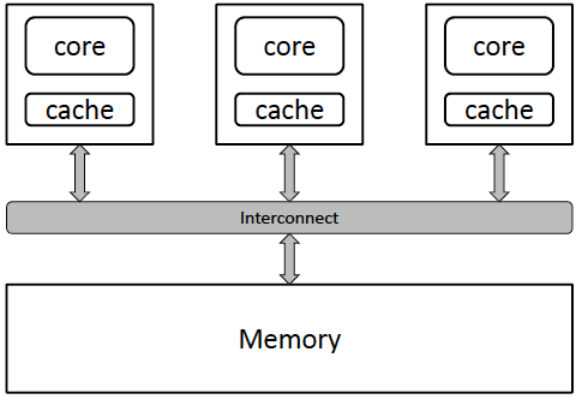
NUMA Model: Non-uniform access times to all words in the memory;

UMA model – Problem solving – Refer to the PDF uploaded

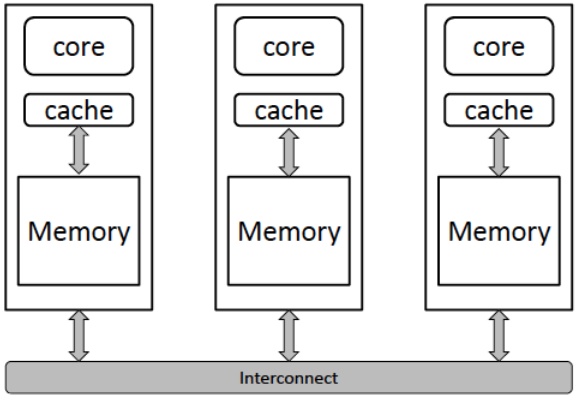
Multiprocessor System Classification

- Memory-based Classification
- Processor based Classification
- Network-based Classification

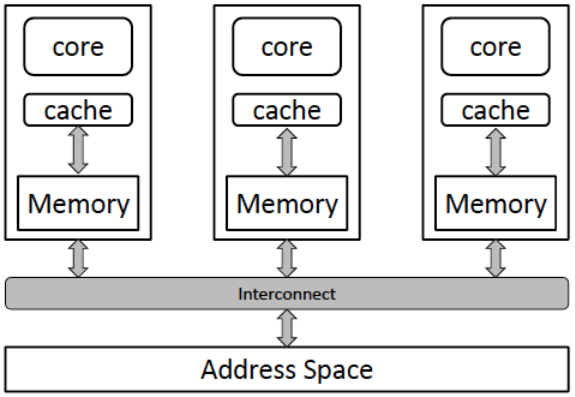
Memory-Based Classification



(a) Shared memory.



(b) Distributed memory.



(c) Distributed shared memory.

Multiprocessor System Classification

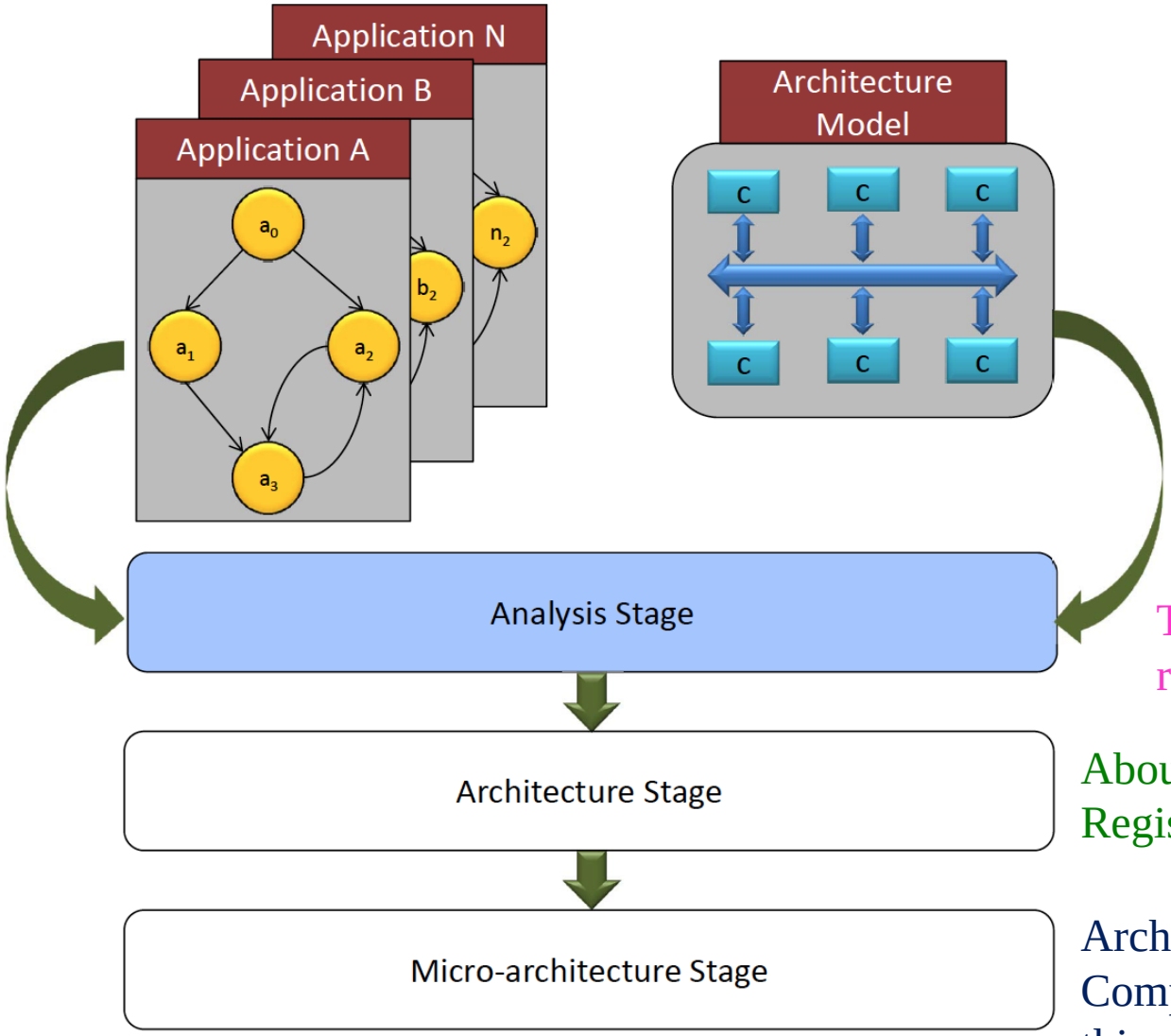
Processor-based Classification

	Homogeneous	Heterogeneous
Advantages	Less replication effort, highly scalable	Application specific, high computation efficiency, low power consumption
Limitations	Moderate computation efficiency, high power consumption	Less flexible, less scalable
Compatibility	data parallelism, shared memory architecture, static and dynamic task mapping	task parallelism, message passing interface, static task mapping
Examples	Lucent Daytona [2], Philip Wasabi [22]	Texas Instrument OMAP [14], Samsung Exynos 5 [24]

Network-based Classification

- Static topology - point-to-point, mesh
- Dynamic topology – physical & logical links change dynamically; Bus, cross-bar (Chapter 2,5)

Multiprocessor System Design Flow



Task Mapping / remapping happens here

About architecture details – Registers, memory etc

Architecture details of all other Components; finally output of this stage is in obtaining register transfer logic (RTL) of the MPs

Supercomputers

A. Vector Supercomputers

A vector computer is often built on top of a scalar processor. Refer to Fig. 1.11 on page 29 (see on our extra slides zone).

If the data are scalar, then it will be directly executed by the scalar unit, and if the data are vector, it will be sent to the vector unit. This vector control unit will supervise the flow of data between the main memory and vector pipeline functional units. Co-ordination is through this control unit.

Two types are described:

(a). Register to register models - CRAY Series

(b). Memory to Memory models - Cyber 205

B. SIMD Supercomputers

An operational model of an SIMD is specified by a 5-tuple

M = <N,C,I,M,R>

where,

N: Number of PEs [Illiac IV has 64 PEs and the Connection Machine CM-2 has 65,536 PEs]

C: Set of instructions directly executed by the control unit (CU) including the scalar and program flow control operations

I: Set of instructions broadcast by CU to all PEs for parallel execution

M: Set of masking schemes; where each mask partitions the PEs into subsets.

R: Set of data-routing functions, specifying various patterns of the INs

PRAM Model (Parallel Random Access Machines)

Theoretical models - convenient to develop parallel algorithms without worrying about the implementation details.

These models can be applied to obtain certain theoretical bounds on parallel computers or to estimate VLSI complexity on a given chip area and other measures even before fabrication.

Time and Space Complexities

The complexity of an algorithm in solving a problem of size s is a function of execution time and storage space.
Time complexity is a function of problem size; Usually worst-case time complexity is what is considered.

If the time complexity g(s) is said to be O(f(s)), if there exists a positive constant c and s_0 such that g(s) <= c f(s), for all non-negative values of s > s_0.

The time complexity function in order notation **O(.)** is the asymptotic time complexity of the algorithm.

Space complexity is similar to time complexity and is a function of the size of the problem. Here, asymptotic space complexity refers to the data storage for large programs.

Deterministic Algorithms - every step is uniquely defined in agreement with the way programs are executed on real computers
Non-deterministic Algorithms - contains operations resulting in one outcome in a set of possible outcomes.

NP-Completeness

An algorithm has a *polynomial complexity if* there exists a polynomial p(s) such that the time complexity is O(p(s)) for any problem of size s.

• The set of problems having polynomial complexity algorithms is called **P**-class problems.

• The set of problems solvable by non-deterministic algorithms in polynomial time is called **NP**-class problems.

Note: P: Polynomial class and NP: non-deterministic polynomial

Now, since deterministic algorithms are special class of non-deterministic class, we can say **P** is a subset of **NP**.

Problems in class P are computationally *tractable*, while the class NP class problems are *intractable*. But, we do not know whether P=NP or P ≠ NP.
This is an open problem for computer scientists.

Examples:

Sorting numbers : O(nlog n)
Multiplying 2 matrices : $O(n^3)$

Non-polynomial algorithms have been developed for traveling salesman problem with complexity $O(n^2 2^n)$. These complexities are exponential. So far deterministic polynomial algorithms have not been found out for these problems. These exponential complexity problems belong to the NP-class.

It is very important to first realize whether a problem is tractable or not. Usually, based on intuition and some preliminary numerical tests, the tractability of the problem will be *felt*. *In practice*, it is very common to reduce a known NP class problem to the problem under study and show that *a problem instance* of the known problem is same as the current problem. Since the known problem belongs to the class of NP, the current problem also belongs to the class of NP.

In the literature, there are some standard problems that are usually mapped on to the problems under study. Largely, familiarity on these known problems is mandatory for anyone attempting to prove that a problem belongs to NP-class.

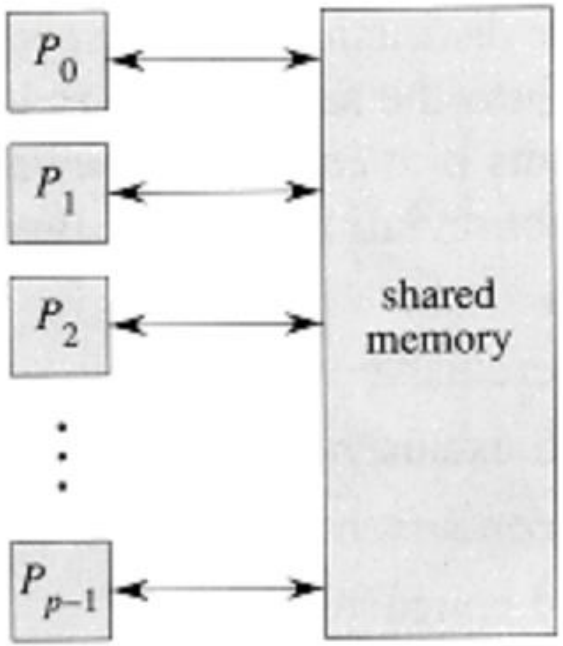
Computers and Intractability - *Theory of NP-Completeness by Gary and Johnson* - a must read book and the bible on this topic.

PRAM model (theoretical) architecture:

PRAM model has been developed by Fortune and Wyllie (1978). This will be used to develop parallel algorithms and for scalability and complexity analysis.

With a shared-memory, 4 possible memory updates are as follows.

- Exclusive Read (ER)
- Exclusive Write (EW)
- Concurrent Read (CR)
- Concurrent Write (CW)



PRAM Variants: Depending on how memory read/write is handled following are the four variants of the PRAM model.

1. EREW-PRAM - Exclusive read and exclusive write

2. CREW-PRAM - Concurrent read and exclusive write

3. ERCW-PRAM - Exclusive read and concurrent write

4. CRCW-PRAM - Concurrent read and concurrent write

In the case of (4), write conflicts are resolved by the following four policies: common, arbitrary, minimum, priority

Useful exercises !

A. Consider a matrix multiplication problem. Let the matrices are of size n x n each.

Analyse the time complexity of the following schemes.

1. A sequential algorithm

2. A PRAM with CREW type

3. A PRAM with CRCW type

4. A PRAM with EREW type

B. Consider multiplying a *n x n* matrix A with a vector X of size *n x 1*. Compare sequential and parallel time complexities. Assume that you have a p processor system. For parallel execution, consider 2 cases when p = n and p < n. Use row partitioning strategy. An all-to-all broadcast communication may be used, if required.

Hint: For an all-to-all communication with a message size of q, the time required is (n.q + ts log p + tw.n), where tw and ts are per-word transfer time and start-up time, respectively.

C. Consider a matrix multiplication problem (2 matrices of size *n X n* each) on a mesh topology.
Demonstrate an algorithm that can perform in **O(n)** time, where $\mathbf{n^2}$ is the number of processors arranged in a mesh configuration.

D. Prefix-sum computation problem. See the "Practice problems" link on the course page. Read the problem statement.

Cost of an algorithm – product of running time and # of processors

Note – The solution described has optimal time complexity among all NON-CRCW machines.
Why? *Because prefix computation* S_{n-1} *takes n additions which has a lower bound of O(log n) on any Non-CRCW parallel machine.*

E. Parallel Mergesort: Demonstrate a parallel version of Mergesort algorithm that runs in O(p) using p processors. Identify the computation and communication steps clearly and compute the complexity.

Some notes on Mergesort for your reading pleasure!

Mergesort is a classical sequential algorithm using a divide-and-conquer method. An unsorted list is first divided into ½. Each ½ is then divided into two parts. This is continued until individual numbers are obtained. Then pairs of numbers are combined (merged) into sorted list of two numbers. Pairs of these lists of four numbers are merged into sorted list of eight numbers. This is continued until one full sorted list is obtained.

Sequential complexity is shown to be O(n log n) for n numbers.

Remarks:

We know that CRCW algorithms can solve problems more quickly than EREW algorithms.

Also, any EREW algorithm can be executed on a CRCW PRAM.
Thus, CRCW model is strictly more powerful than EREW model.
But, how much more powerful is it?

It has been shown that a p-processor EREW PRAM can sort p numbers in ***O(log p)*** time. So, can we derive an upper bound on the power of a CRCW PRAM over an EREW PRAM?

Theorem: A *p-processor* CRCW algorithm can be no more than O(log p) times faster than the best p-processor EREW algorithm for the same problem, i.e., $T_{CRCW} \leq O(\log p).\ T_{EREW}$

Speed-up: Time (single processor) / Time (n processors)

If F is the fraction of a program that cannot be parallelized or concurrently executed then, speed-up S(n,F) with n processors is given by,

$S(n,F) = Ts /[(F.Ts + Q]$, where $Q = (1 – F)Ts / n$

(Ts is the time to run in serial fashion)

This expression is known as *Amdhal's Law*.

Q: Plot S(n,F) with respect to n and F and interpret the behavior. What happens to S(n,F) when n tends to infinity?

Leisure reading! Read an interesting article about another form of Amdhal's law via your Useful Links zone!

To be cont'd…