Chapter 1. Introduction to Multiprocessor Systems

Contents of this Chapter

- Overview of Computer generations hierarchy
- Elements of Modern Computer Systems
- Flynn's Classification
- Performance Issues
- Programming for Parallelism

- **Multiprocessors and Multicomputers**
- **Case Study: UMA, NUMA, COMA models**
- **Vector Supercomputers**
- **SIMD Supercomputers**
- **Introduction to PRAM and VLSI models**
- **Architectural development tracks**

Reference *: Kai Hwang - Chapter 1 pages 3-46*

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;

Feel the intensity of computations (pain!!?) first!!

☹ "Ugh!"

(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!

Size, Nature and Characteristics of Data

Data – Specific format - Prepared for each climate trial period (usually, 30/40 years, few tera bytes).

Data contain a set of variables in 4 dimensional arrays (time, vertical-level, latitude, longitude).

Raw Data Size – Few 100s of GB for each month.

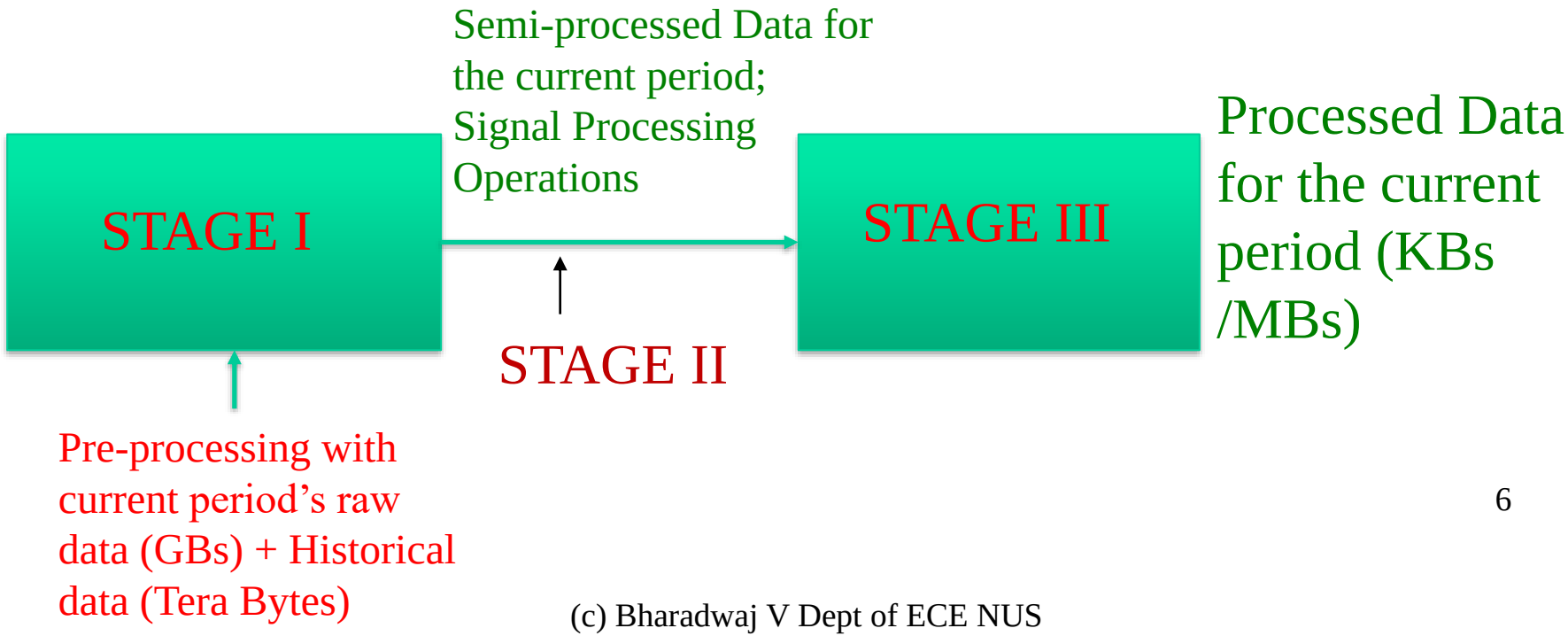
Stage-wise Processing – Involves Three stages in a pipelined fashion

5

Types of data

- It is possible that the data may "expand" or "shrink" during processing!

Weather prediction data processing: Three stages: Stage I (during pre-processing)- raw data + historical data; Stage III (tail end processing) – Processed data (useful output) in KBs/MBs



First Stage: Raw data is processed to generate several GBs new set of data for each Calendar year + historical data (TBs);

Second Stage: Use the first stage results to create the input and boundary conditions for the main computation. The storage requirement for each calendar year of second stage is 100s of GB.

Third/Output Stage: Use the generated data from Stage 2 for the actual computation for 1 Calendar Year is generated!

What about storage requirements?

Estimated Total storage for the whole process ~ 10TBs ☹

What about $ cost?

Estimated Total $ cost for the whole process ~ $2M for a 48 node multi-core cluster system ☹

(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!!

Rubik Cube….

Data for Time in hours Vs Number of processors required to solve

Algorithm	**Time required to solve the Cube in hours**
Sequential	51.94972
np=2	43.63417
np=4	40.55639
np=6	27.87778
np=8	22.23417

np= Number of processors used in the parallel implementation.

Computer Generations hierarchy

- **Refer to Table 1.1 - page 5** ***(** outdated stuff!! But still ok to appreciate the trend)***

Current day technology trends:

IC logic technology – Transistor density increases about 35% per year; increase in die size is less and slower; combined effect of transistor count growth rate on a chip is about 55% per year; device speed scales more slowly;

Last decade breakthrough architecture – 2007 IBM Cell - ~250 million transistors!!

Semiconductor DRAM – Density increases by 40% – 60% per year; Cycle time has improved slowly, decreasing about 1/3rd in 10 years; Bandwidth per chip increases about twice as fast as latency decreases; interfaces to DRAM have also improved the bandwidth;

1978 – ’81:	16K -	$10 per Dram chip
1984 – ’89:	256K -	$38 - $6 per Dram chip
1986 – ’91:	1M -	$40 - $6 per Dram chip
…		
1993 – ’98:	64M -	$74 - $4 per Dram chip
1999 –’2001:	64M -	< $10 per Dram chip

20+ yrs trend

……..

Magnetic Disk technology – Disk density is improving more than 100%; quadrupling in two years; prior to 1990 density increased about 30% p.y, doubling every 3 years; Access time has also improved by 1/3rd in 10 years;

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 trend?

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

Walking with Computing Power!! - PDAs, 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 centres!!

Hardware ASICs - 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.

Intel CPU Microarchitectures – Generations hierarchy

		Number of pipeline stages	Max.clk	
2015	Skylake	14 (16 with fetch/retire)	4200 MHz	
2016	*Goldmont*	20 unified with branch prediction	2600 MHz	14nm
	Kaby Lake	14 (16 with fetch/retire)	4500 MHz	
2017	Coffee Lake	14 (16 with fetch/retire)	5000 MHz	
	Goldmont Plus	? 20 unified with branch prediction ?	2800 MHz	
2018	Cannon Lake	14 (16 with fetch/retire)		10 nm
	Cascade Lake	14	? MHz	14 nm
	Whiskey Lake			
2019	Ice Lake			10 nm

Intel's Cannon Lake Processor (2018) innovative aspects: Transition from 14nm to 10nm technology Transistor density of ~100 mega transistors per mm^2 (2.7X increase over 14nm tech); Uses 3rd generation FinFET Technology

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

Intel CPU Microarchitectures – Generations hierarchy

Other variants (Intel chips) after 2019 till date uses 14nm to 7nm (Willow cove, Golden Cove, etc)

News from Intel!

- May 2020 - 11th Gen Tiger Lake H-series (already available on laptops)

- May/June 2021 – Two new U-series chips - one of which marks the first low-voltage 5.0GHz clock speed!!! Base clock is ~3GHz but it can stretch up to 5GHz;

- Under 5G platform Intel jointly with Qualcomm released "Intel 5G solutions" for modems in their laptops with M.2 card; These cards enable 5G modems to be connected to desktops/laptops;

Moore's Law & its hype…

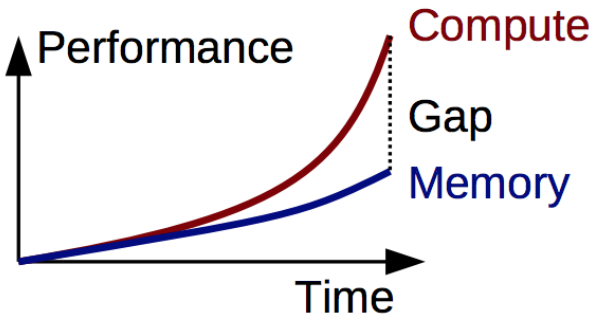
Moore's Law is a computing term which originated around 1970; the simplified version of **this law states that processor speeds, or overall processing power for computers will double every two years**.

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

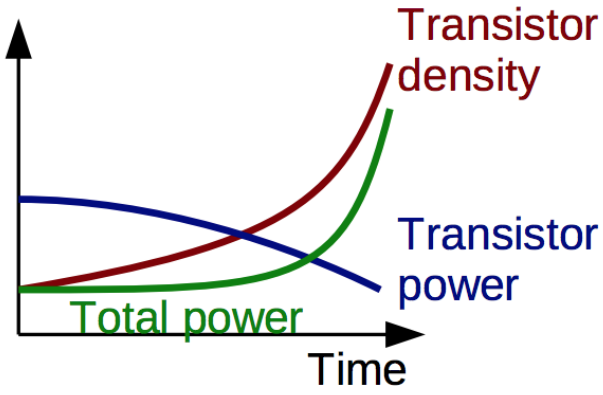
Comparing Computing Power - Your mobile vs. Supercomputer!

- 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!

On the whole, tech trend …



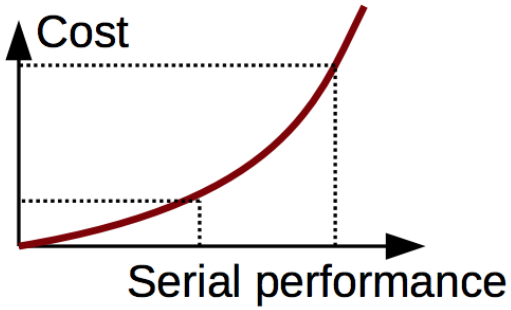
Memory performance



Power performance

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

Consequently…



ILP Performance

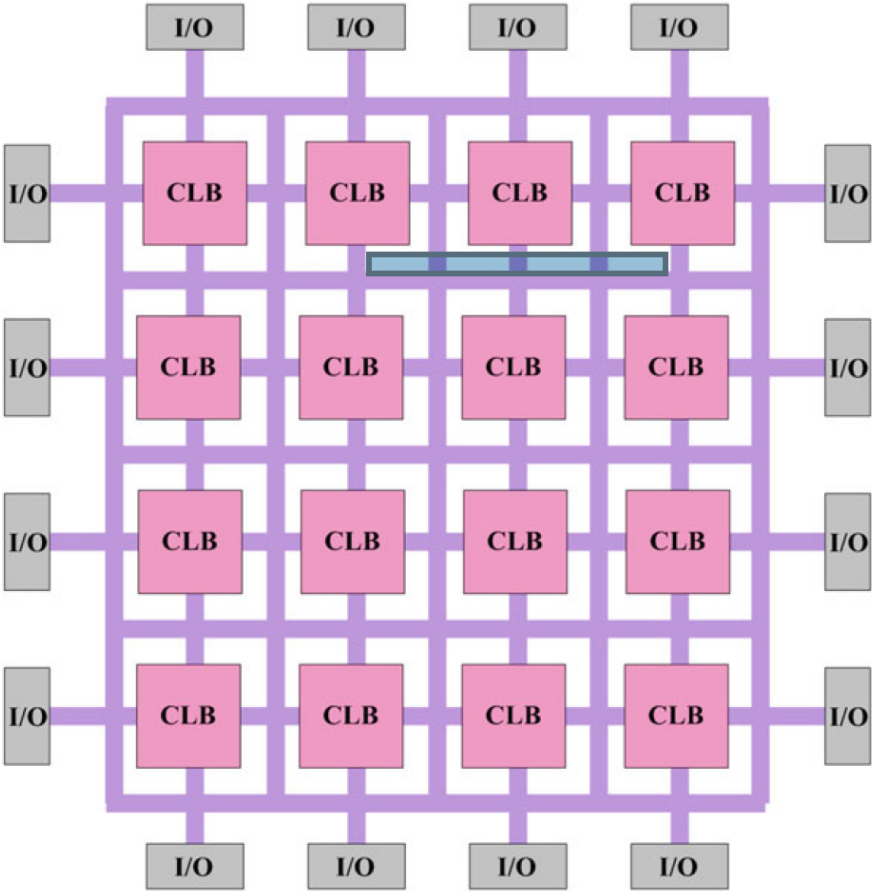
Two approaches

(A) Flexible hardware-cum-programming

(B) Hardware acceleration

(A) Flexible hardware-cum-programming

FPGA Architecture



Normally FPGAs comprise of:

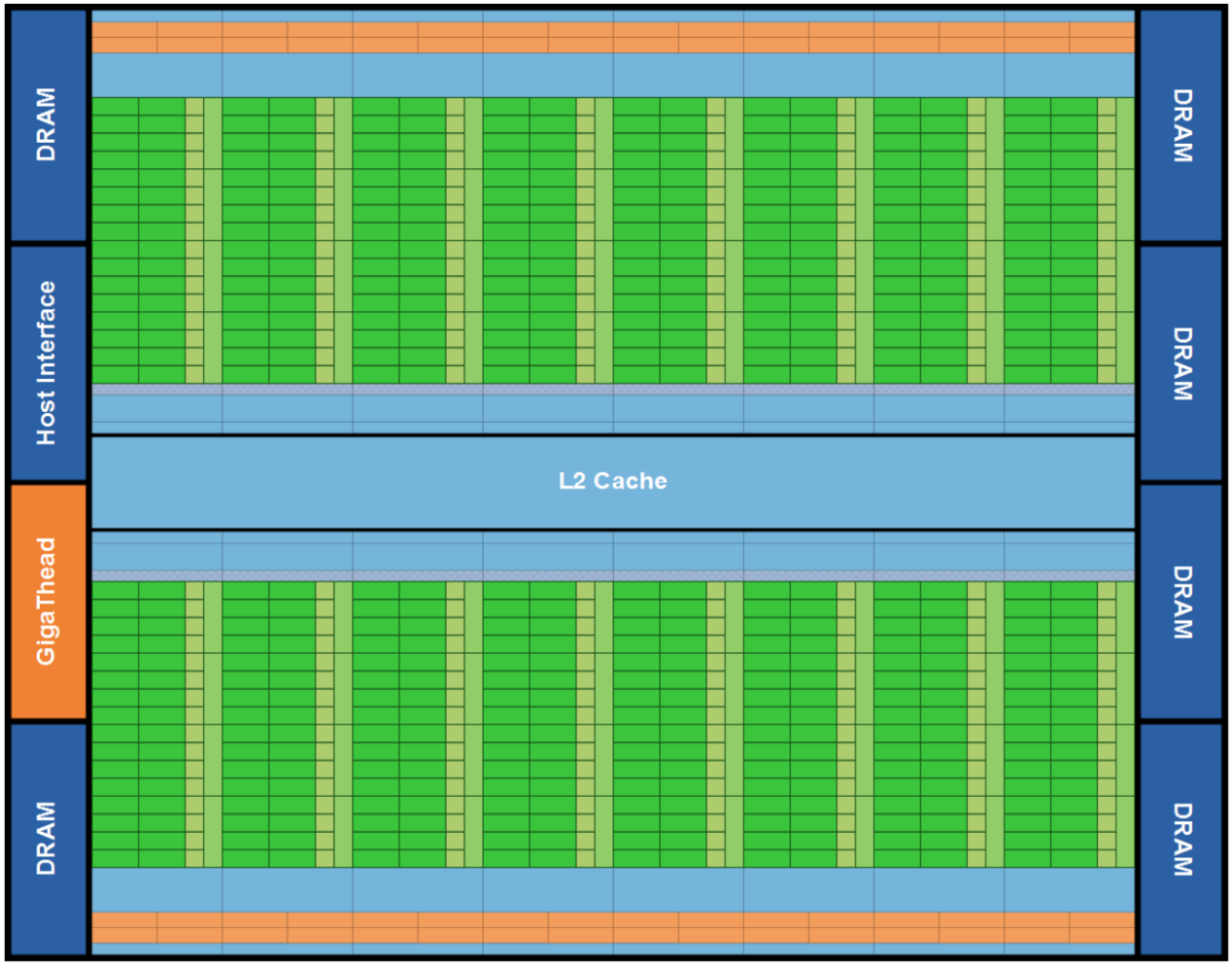
- Programmable logic blocks which implement logic functions.
- Programmable routing that connects these logic functions.
- I/O blocks that are connected to logic blocks through routing interconnect and that make off-chip connections.

Some FPGA specs available in the market…

FPGA	Xilinx Spartan-3E XC3S100E (Basys2-Board)	Xilinx Virtex-6 XC6VLX130T (SRU Board)	Xilinx Virtex-7 XC7V2000T	Xilinx Virtex „UltraScale“ XCVU440
I/O lines	83	600	1200	1456
Logic cells	2,160 (4-input LUT)	128,000 (6-input LUT)	1,954,560	4,407,480
Internal Block RAM memory	72 kBits	9,500 kBits	46,512 kBits	88,000 kBits
Price (@1 pcs) www.digikey.com	8.47€	817.39€	16,286.94€	???

More details on CLBs, routing, etc., in Chapter 3, if time permits…

GPU - FERMI CUDA Architecture



3.0 billion transistors, features up to 512 CUDA cores

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 3! Stay tuned!)

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

CUDA cores are parallel processors similar to a processor in a computer, which may be a dual or quad-core processor. GPUs, however, may have several thousand cores.

GPU	G80	GT200	Fermi
Transistors	681 million	1.4 billion	3.0 billion
CUDA Cores	128	240	512
Double Precision Floating Point Capability	None	30 FMA ops / clock	256 FMA ops /clock
Single Precision Floating Point Capability	128 MAD ops/clock	240 MAD ops / clock	512 FMA ops /clock
Special Function Units (SFUs) / SM	2	2	4
Warp schedulers (per SM)	1	1	2
Shared Memory (per SM)	16 KB	16 KB	Configurable 48 KB or 16 KB
L1 Cache (per SM)	None	None	Configurable 16 KB or 48 KB
L2 Cache	None	None	768 KB
ECC Memory Support	No	No	Yes
Concurrent Kernels	No	No	Up to 16
Load/Store Address Width	32-bit	32-bit	64-bit

(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!

Top Supercomputers (June 2020 listing)

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,299,072	415,530.0	513,854.7	28,335
2	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148,600.0	200,794.9	10,096
3	**Sierra** - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94,640.0	125,712.0	7,438
4	**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
5	**Tianhe-2A** - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000, NUDT National Super Computer Center in Guangzhou China	4,981,760	61,444.5	100,678.7	18,482

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

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

The system of elements is shown in the following figure.

Refer to Fig 1.1. on page 7

- **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 - Fig. 1.2**

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

What can we say on the volume of data to be handled?

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 Fig. 1.4 on page 12

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! $T^* = \min \Sigma T_k$, $k=1,2,..,n$

In a multi-programmed environment, I/O and system overheads may be allowed to overlap with CPU times required in the other 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*

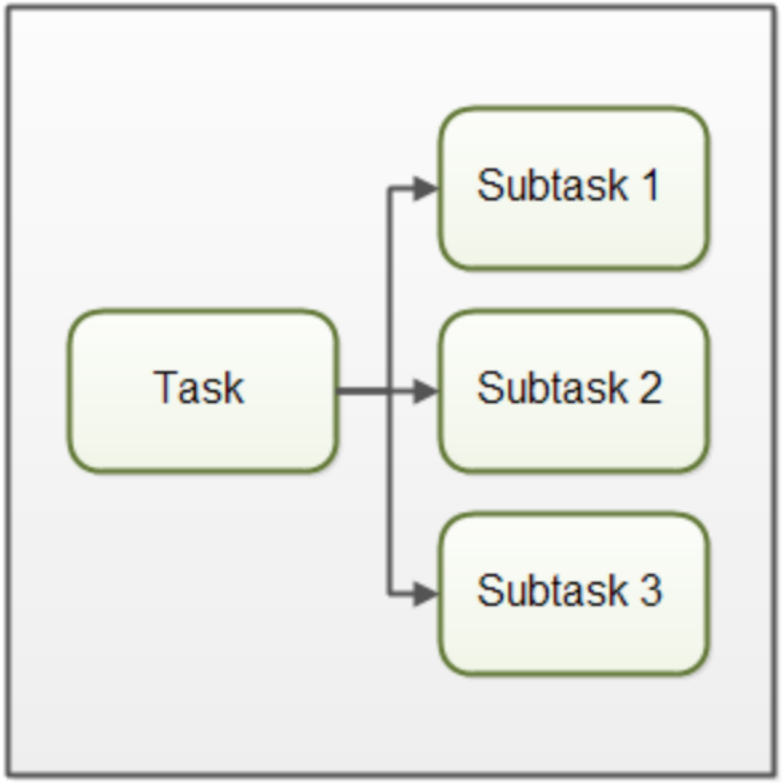
Explicit Parallelism

- *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 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

(See Fig. 1.10- page28)

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.

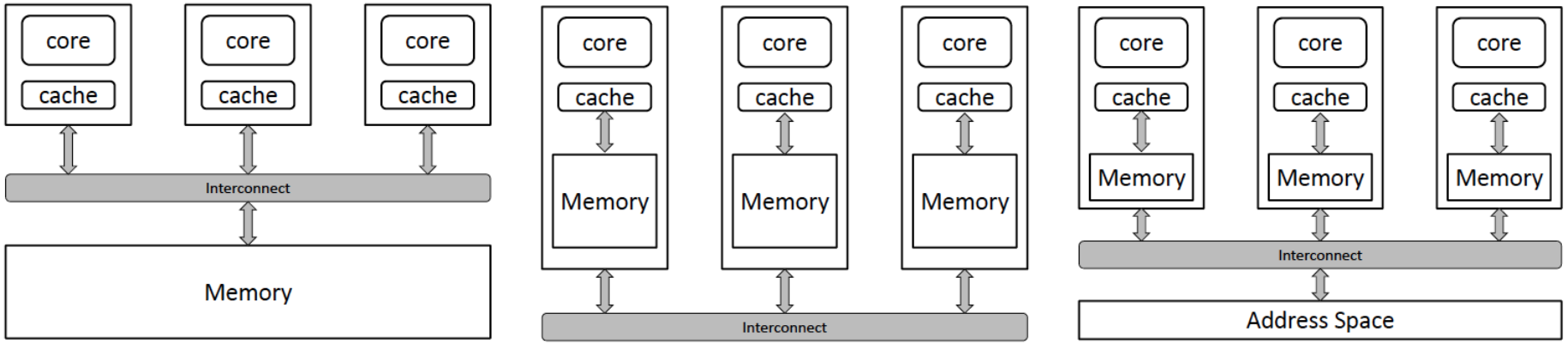
NUMA Model: Non-uniform access times to all words in the memory;

Reading Assignment: **Pages 23-27(from COMA till before Section 1.2.3)**

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

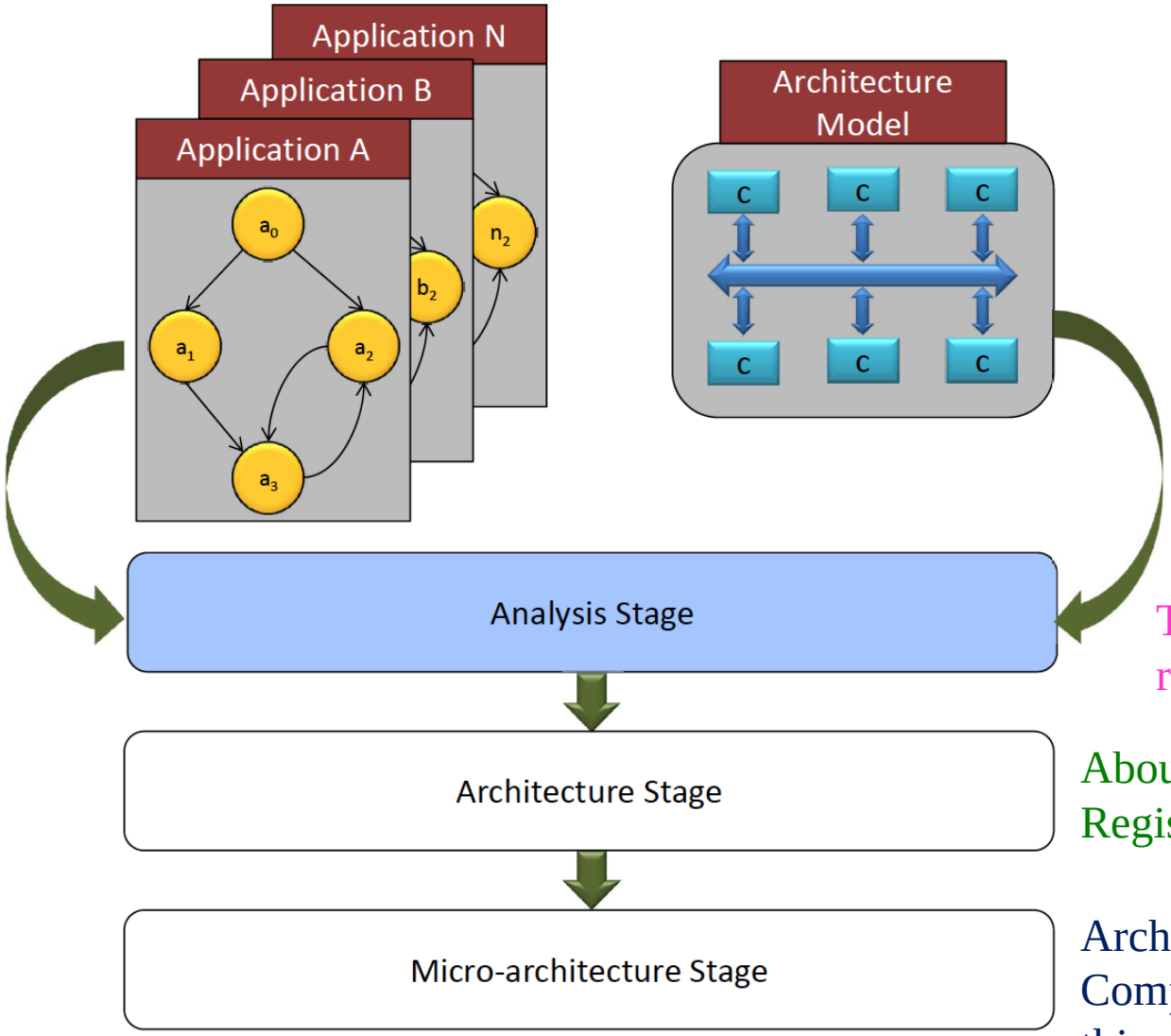
Processor-based Classification

	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.

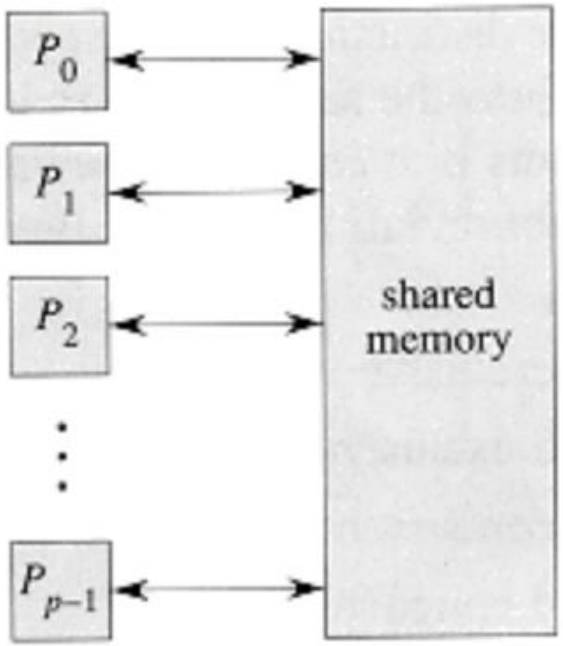
Computers and Intractability - *Theory of NP-Completeness by Gary and Johnson* - a must read book and the bible on this topic.

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!

Embedded Processors - *Some remarks*

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

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

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

$$F(f) = (V_T + f/2k) + sqrt\,((V_T + f/2k)^2 - V_T^2)) \quad (2)$$

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

$$P = 0.5 \,.f.C.V_{dd}^2 = 0.5 \,.\, F.\, C.\, F(f)^2 \qquad (3)$$

(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).

VLSI Model

READING ASSIGNMENT

Parallel computers rely on chip fabrication technology to make arrays of processors, switching networks, etc.

AT^2 Model (Clark Thomson 1980)

A: Chip area; T: Latency for completing a given computation using a VLSI circuit chip

s: Problem size involved in the computation

Reading Assignment: Pages 38 -40, 41(Section 1.5)-45

Good luck!

Some useful references on Rubik cube solution approaches and algorithms…

http://jeays.net/rubiks.htm
http://www.cs.princeton.edu/~amitc/Rubik/solution.html
http://www.rubiks.com, Rubik/ Seven towns, 1998
http://home.t-online.de/home/kociemba/cube.htm
http://www.math.ucf.edu/~reid/Rubik/
http://www.math.rwthachen.de/~Martin.Schoenert/CubeLovers/Mark_Longridge__Superflip_24q.html