# CHAPTER #1 "WHY PARALLEL COMPUTING"

## "WHY DO WE NEED EVER-INCREASING PERFORMANCE"

### Role of Computational Power In Advancements:

- Computational power has driven significant advances across diverse fields such as science, Internet and Entertainment
- Examples include decoding the human genome, accurate medical imaging, fast Web searches, and realistic computer games.
- Future advancements often depends on previous increases in computational power, making continous improvement essential.
- As computational power grows, so does our ability to tackle increasingly complex problems.

## "Examples of Problems Required Increased Computational Power"

### 1-) Climate Modeling:

More accurate models are required to understand climate change, requiring the inclusion of interaction b/w atmosphere, oceans, land and polar ice caps. Detailed studies of potential ~~invest~~ interventions on global climate requires great computational resources.

.) Protein Folding:

Misfolded protein are believed to play a vital role in diesreases like Huntington's, Parkinson's and Alzheimer's. Current computational limits Current computations limits our ability to study such complex molecular configuration as it requires more power for research.

.) Drug Discovery:

Increased computation powers enables more effective research into medical treatment. Analyzing genomes for alternative treatment requires extensive computing analysis especially for diesases where known treatments are ineffective for certain individuals.

.) Energy Research:

Enhanced computational models for wind turbines, solar cells and batteries can lead to more efficient clean energy resources.

.) Data Analysis.

Global data volume is doubling every two year which is useless without proper analysis. Fields like genomics, particle physics, medical imaging, astronomy requires complex analysis on vast amount of data requiring very powerful computational power.

# "Why We Are Building Parallel Systems"

Problem.

- The size of transistors (electronic switches) on integrated circuit were decreased in order to increase the overall speed of the integrated circuit.
- But it requires more power and most of the power is dissipated as heat which became unreliable.
- To tackle it the transitor density was started to be increased but it wasn't permanent solution.

Solution:-

.) Instead of building complex single processor the industry shifted to parallelism.
.) This approch involves placing multiple, relatively simple processors (or cores) on a single chip, leading to development of multicore processors.

# Why We Need To Write Parallel Program

.) Most existing programs were written for single-core system and can not utilize multiple cores effectively
.) Running multiple instances of the same program in multicore system is often useless i-e running a game multiple time. A game should run faster with better graphics

Need:-

.) Serial programs must be rewritten as parallel program or converted by translation program that automatically transforms it into parallel program.
.) However, it's a challenging task and scientist have found limited success to automate the process especially in C/C

## Challenges:-

- ) Although certain serial programs can be converted into parallel program but it's inefficient or useless
- ) For example matrix multiplication or Summing n no. of numbers.
- ) This is bcz the computation depends on previous result so it follows a serial approch and all resources of multicore system can't be used efficiently

```
for (i=0 ; i<n , i++) {
        sum += i++;
}
```

## Rewritting code for Parallelism

- ) Bcz of such problems the whole algorithm as to be rewritten with a new and efficient logic
- ) In parallel approach the task could be done by giving each core a part of partial sum which are combined into final sum. It makes it more efficient

Parallel code:

// Each code get a partial sum

```
my_sum = 0
my_first_i = ____ ;
my_last_i = ____ ;
```

} get partial index to sum

~~my_A = 0;~~

```
for (int my_i = myfirst_i ; my_i < my_last_i ; my_i++){
```

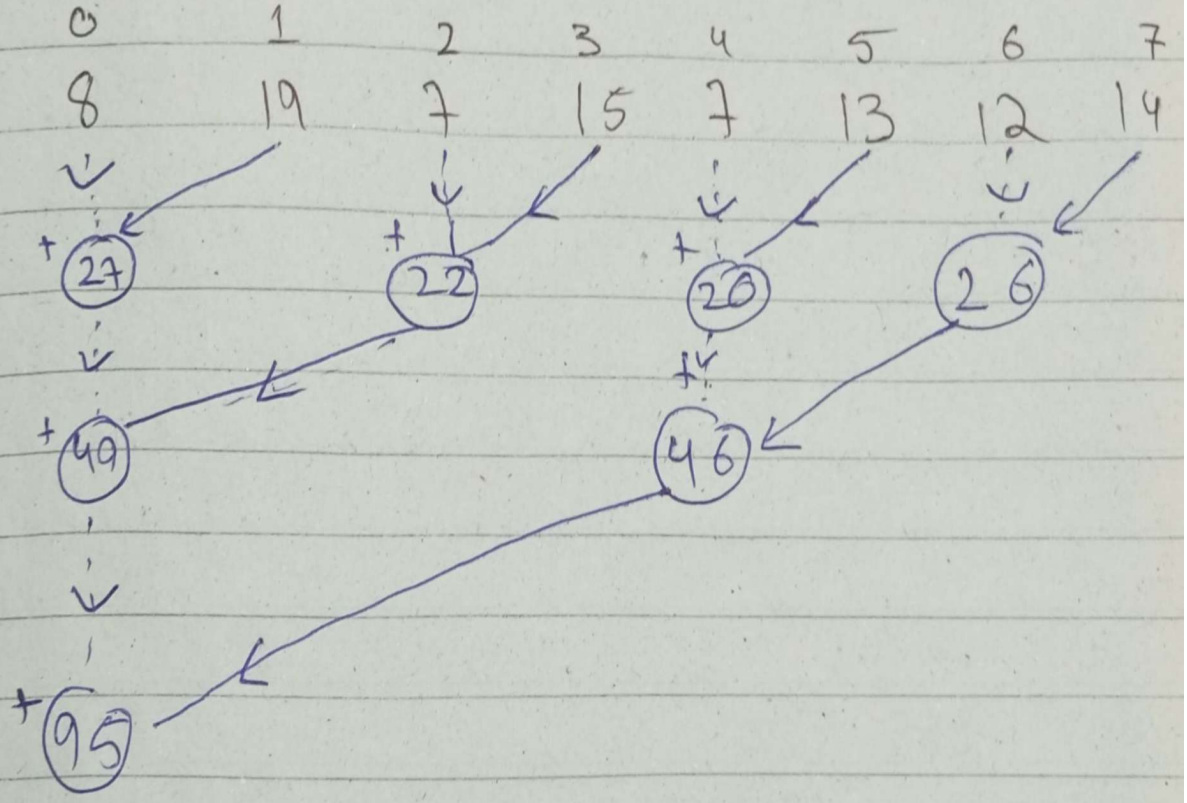
~~my_A = my_i~~

```
    my_sum += my_i;
}
```

// A core combining all partial sum into 1

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

Consider master core 0 adds 8 + 19 + 7 + 15 + 7 + 13 + 12 + 14 = 95




This is how partial sum is added to whole sum.

# How Do We Write Parallel Program

The basic idea is of partioning the work to b done among cores. There are 2 widel used Approaches:

1-) Task Parallelism

Different cores are assigned dif tasks that togheter solve a computin problem. Each core works on a unique part of problem, performing differeat operations or funtions.

It's usefull when task can be divided into distinct tasks that dont necessarily operate on same data.

For Example:

Consider a professor (Prof P) with 100 stude and 4 TAs. After final exam Prof P and 4TA need to gode the exam. They will do it by

Prof 1 grades Qs1
TA 1 grades Q22
TA 2 grades Qs 3
and so on.

Here the Prof & and 4 TAs are the cores and Question are the task being graded by them.

## 2-) DATA PARALLELISM :-

The data is partioned among the cores and each core performs the same operation on it's portion of the data.

This is Useful when the problem involves a large dataset that can be divided into smaller, independent pieces.

For Example,
Alternatively, the Prof & and 4 TAs divide the piles of 100 exam paper and distribute 20 to each so each person is doing the same work on ~~the~~ it's portion of data

## "Example From Global Sum Problem"

Data-Parallelism: The first part involved each core computing the sum of a subset of a value. This is data-parallelism. Each core perform the same operation on a different portion of data

Task-Parallelism: The second part involves one core (the master) receiving and adding the partial sum from other cores, while the other cores sent their result to the master. This is example of task-parallelism.

## "Coordination In Parallel Program"

Parallel programs often require coordination among cores, which can make them more complex to write. This coordination involves

i) Communication: Cores need to share data and exchange data i.e in the global sum problem, cores send their partial sum to master core

ii) Load Balancing. The work should be evenly divided among cores. If one core has more works than others, it becomes a bottleneck, wasting the processing power of the idle core.

iii) Synchronization: Cores may need to wait for others to reach a certain point before proceeding. For example if one core is initializing data that other cores will use, they must synchronize to ensure the data is ready before they begin processing.

```
if (I'm master core)
    for (my_i = 0; my_i < n; my_i++)
        scanf("%lf", &x[my_i]);

Synchronize_core();
    // for loop must wait for master core to finish analysing data //.
for (my_i = my_first_i; my_i < my_last_i; my_i++)
    my_sum += x[my_i];
```

NOTE: Most powerful Parallel Programs are written on C, C++. Higher Level Lang make writing Parallel Program easy but often sacrifice performance.

# "LEARNING PARALLEL PROGRAMMING

To Explicitly write parallel program using the C language we require 3 extensions

.) Message-Passing Interface (MPI):-

TYPE:- Library of type definitions, functions and macros

PURPOSE:- Designed for programming distributed-memory system where each core has it's own private memory. Cores communicate by sending messages through a network.

.) Pthreads (Posix Threads):-

TYPE:- Library of type definitions, functions and macros

PURPOSE:- Designed for programming shared-memory system where all cores share access to computer's memory. Pthreads provide a mechanism for accessing and coordinating shared memory.

.) Open Mp:-

TYPE:- A combination of library and modification to 'C' Compiles.

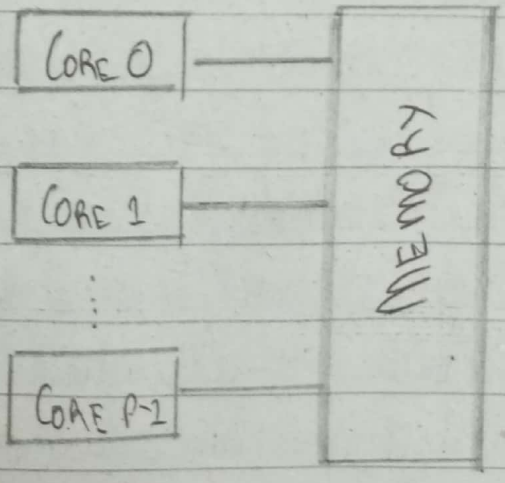
PURPOSE:- Also designed for shared memory system, but it's higher-level extension compared to Pthreads as it allows simpler parallelization of code, often with a single directive.

Each of these Extensions serves different purpose and are designed for different types of parallel computing systems.

There are 2 types of Parallel System:

1-) SHARED MEMORY SYSTEM:-

- In these systems cores can share access to the computer's memory. This means that each core can read and write to the same memory locations. Coordination in shared memory system involves examing and updating these shared memory system.
- Pthreads and OpenMP are designed for it. While Pthreads offer fine-control with detailed coordination constructs, OpenMP simplifies parallelization sometimes with a single directive.



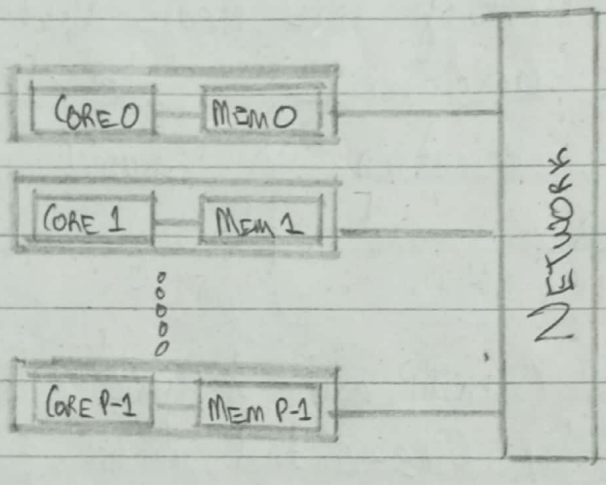

NOTE

#pragma omp parallel for reduction (+:sum) it's a directive which breaks the loop of summing n nos into all cores.

Directive: is a special inst given to compiler. It tells compiler how to process certain parts of the code.

## 2-) Distributed Memory System:

- In this each core has it's own private memory, and cores must communicate explicitly often by sending messages across a network.
- MPI is designed specifically for it as it provides the necessary mechanism for message passing between cores.




## "Shared Memory Extensions:- Pthreads vs OpenMP"

OpenMP: A high level extension that allows easy parallelization for loops and other constructs i.e a simple addition loop can be parallelized with a single MP directive.

Pthreads: Offers more detailed control over parallelism. It provides coordination that OpenMP might not support. It makes certain complex parallel programs easier to manage.

# CONCURRENT COMPUTING:-

- Multiple Tasks can be in progress simultaneously. This doesn't mean that the task are executed at the same exact time, but rather they can be started, run and completed in overlapping time period.
- For example various processes like running a web browser, playing music and editing a document can be in progress at the same time, even on a single-core machine

# PARALLEL COMPUTING:-

- It involves multiple task cooperating closely to solve a problem. The task run simultaneously on different cores of the same computer. They often share memory or connected through high-speed network
- For example running multiple computation at the same time on a multi-core processor to speed up the processing of a large data set.

# DISTRIBUTED COMPUTING

- It involves multiple tasks that may need to cooperate with other tasks or programs located on different computers
- They are often seperated by significant distances and the tasks are typically loosely coupled i.e they don't necessarily need to interact closely or frequently.
- For example a web search engine where different servers are located around the world. It processes different part of the query when required then combine the result.

## "KEY POINTS"

Concurrent vs Parallel: All parallel distributed programs are concurrent but all concurrent programs are not parallel i.e a multitasking OS is concurrent but not parallel.

Parallel vs Distributed: Parallel tasks are usually executed on cores that are physically close and share memory whereas distributed task are executed on seperate machines that are far apart and connected by a network.

CHAPTER # 2 "P

VON N

A foundation
a single s
and instructi

MAIN COMPO

- Main Me
  both
- Central
  perfo
- Interconnec
  trans
  CPU

- Main
  locations e
  access the
  divided
  fixed siz
- CPU sue
  directin
  I/O de