

# Introduction to Parallel Computing

*Bootcamp for SahasraT*

*7th September 2018*

*Aditya Krishna Swamy*

[adityaks@iisc.ac.in](mailto:adityaks@iisc.ac.in)

*SERC, IISc*

# Acknowledgments

- Akhila, SERC
- S. Ethier, PPPL
- P. Messina, ECP
- LLNL HPC tutorials

# What is scientific computing?

- "Computational Science" - 3rd paradigm
- "Scientific Computing" - encompass engineering computations as well as the classical scientific applications such as computational physics, chemistry, and astronomy
- Included in "computing" are the associated systems of networks, visualization, and data storage and retrieval
  - not just the numerical computation aspects
- Scientific computing is an experimental endeavor, the use of real systems is essential, it has an engineering flavor
- Often the first experiments with a new approach are crude but they serve to identify what works
  - At that point it is possible to abstract the key features and devise a better design AND implementation; the latter steps may be taken by others

# In scientific computing there is always demand for more speed, capacity

- Computational scientists have an insatiable need for more powerful computers (faster, able to handle more data)
- Resolution (finer grid, more atoms)
- Dimensionality (1-D, 2-D, 3-D, etc.)
- Complexity (multi-physics, multiple time and spatial scales)
- Fidelity (equations, geometry, realistic conditions and dimensions)
- Time to solution
- Algorithmic complexity, e.g.,  $O(N^7)$

# High Resolution Hurricane Studies

*Greg Holland, NCAR*

Sample results from simulation of 2005 hurricane season.

These data are freely available to download from NCAR, useful for research

on the role resolution plays in representation of individual weather events and on seasonal statistics

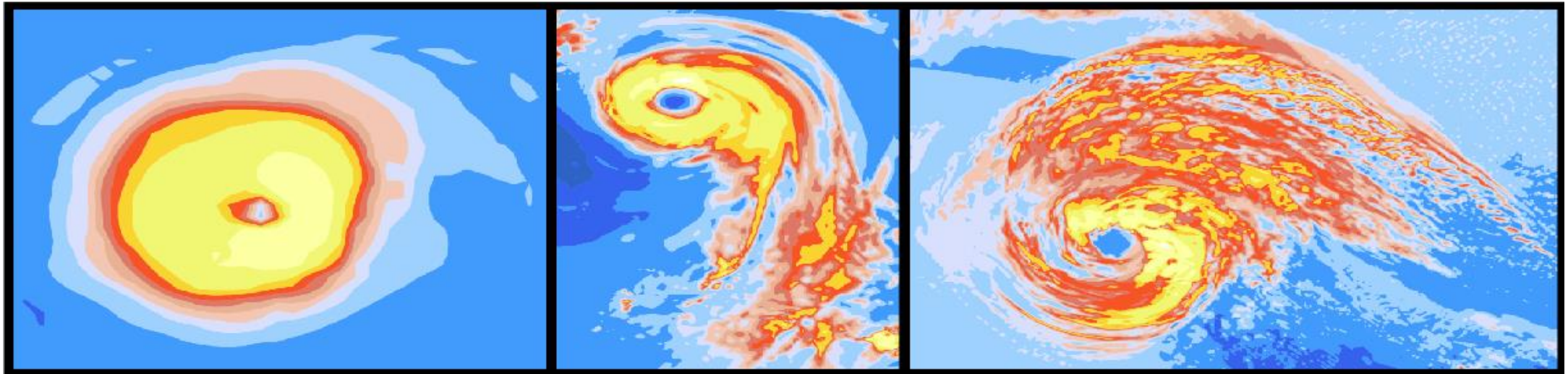
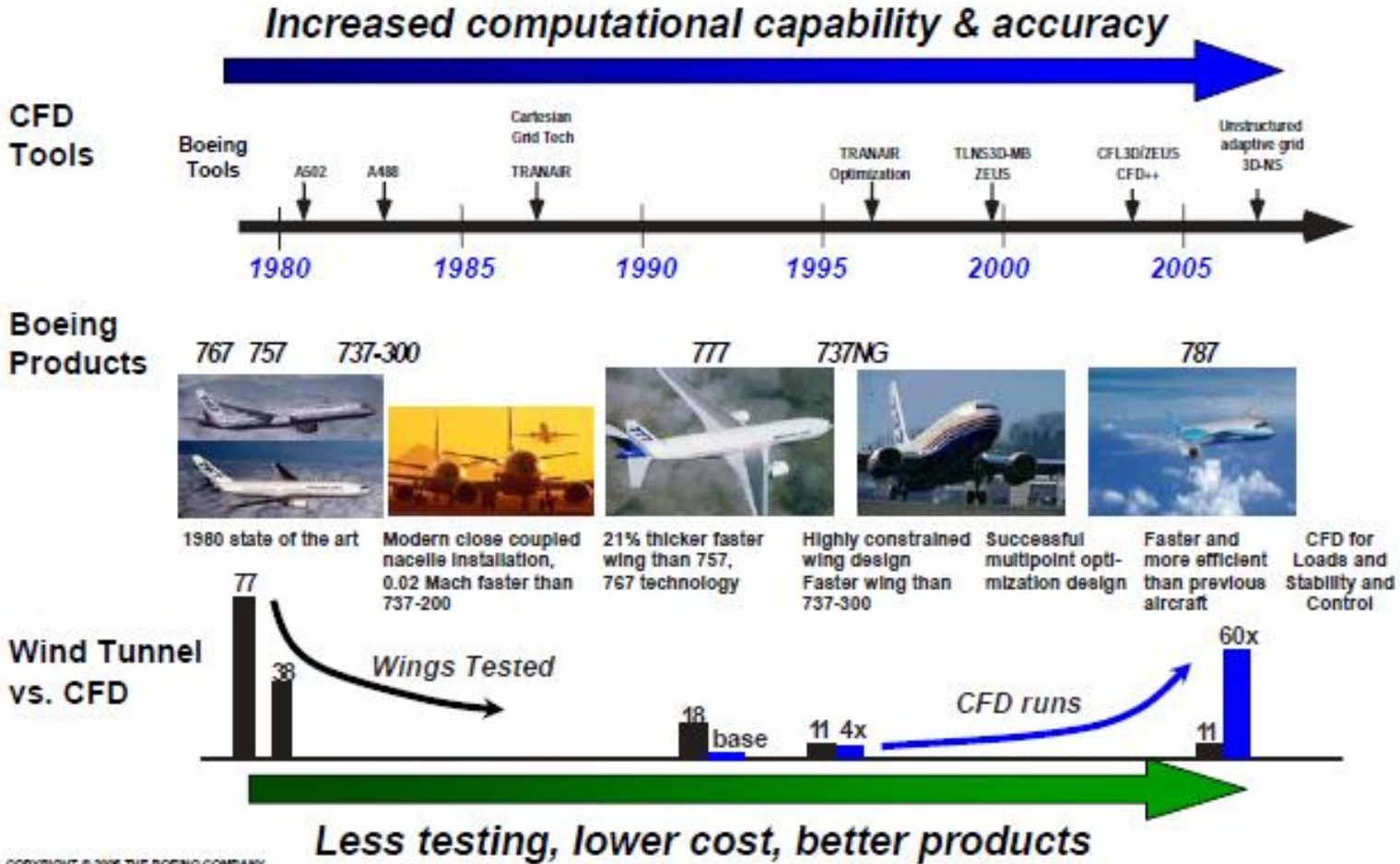


Image : Strongest hurricane (not necessarily the same) produced per simulation at 36km, 12km and 4km.

# CFD Has Significantly Improved the Wing Development Process

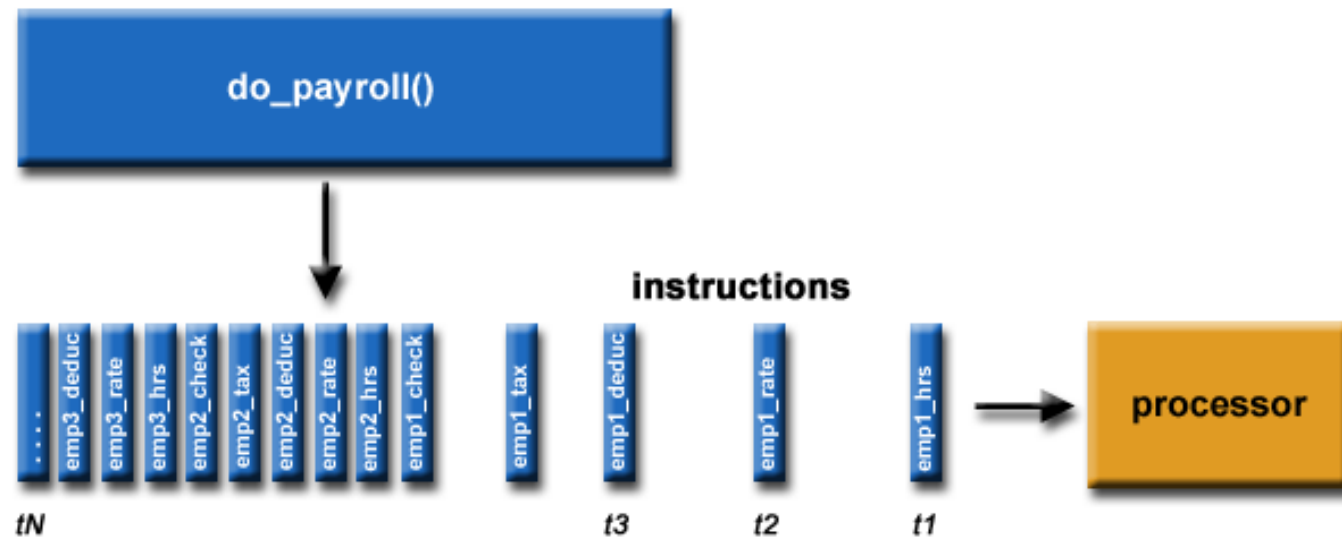


# Scientific computing requires broad expertise *in addition to the research domain*

- Knowledge of computer architectures, mathematical models and numerical algorithms
- Proficiency in programming methodologies and languages
- Software architecture, debugging and performance measurement tools, visualization
- Software engineering for working in teams
- Methods for building “community codes”
- Methodologies and tools relevant for data-intensive applications
- Frameworks for scientific workflows

# Serial Computing

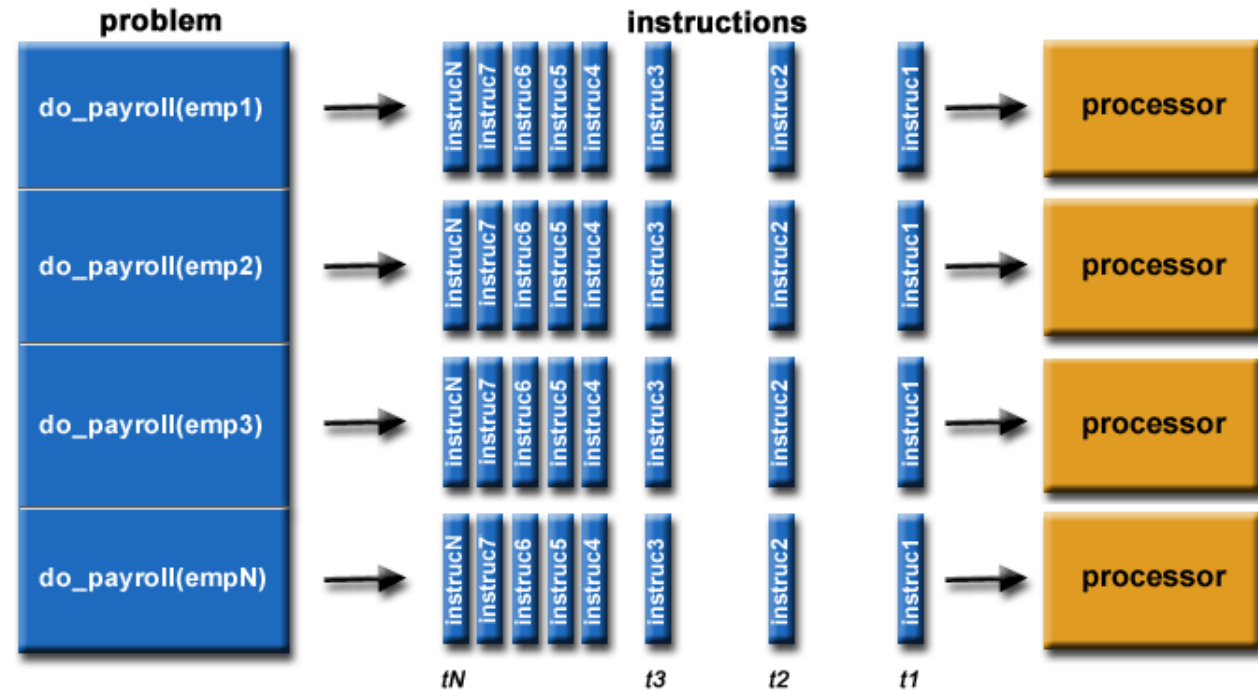
- Traditionally, software has been written for serial computation
- A problem is broken into a discrete series of instructions
- Instructions are executed sequentially one after another
- Executed on a single processor
- Only one instruction may execute at any moment in time





# Parallel Computing

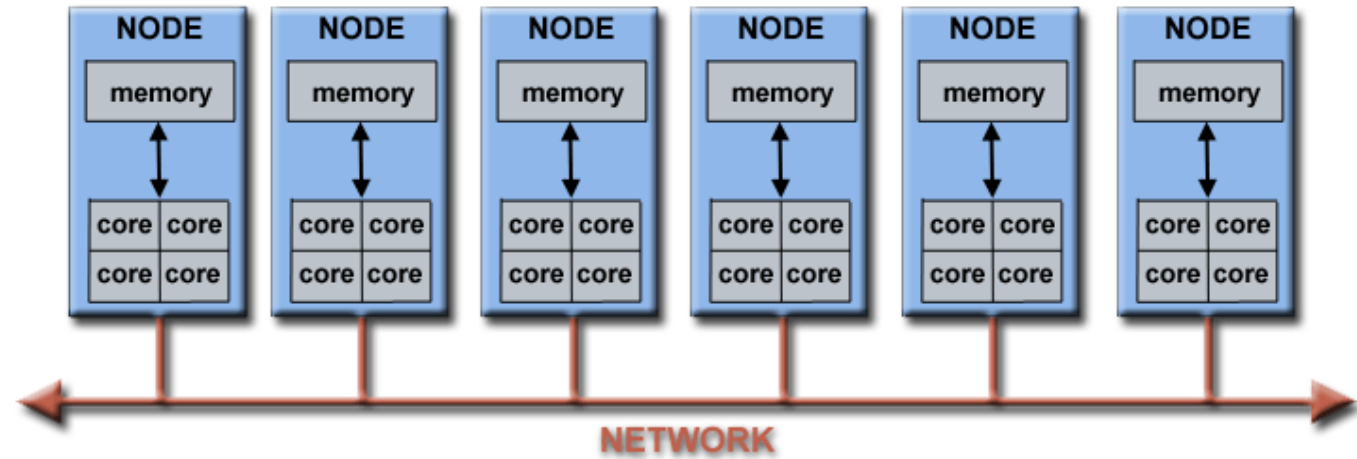
- Simultaneous use of multiple compute resources to solve a computational problem.
- Run on multiple CPUs
- Problem is decomposed into multiple parts that can be solved concurrently.
- Each part is decomposed into a set of instructions.
- Instructions are executed simultaneously on different CPUs



# Parallel Computer Architecture

## Compute Resources

- Single Computer with multiple processors.
- A number of Computers connected by a network.

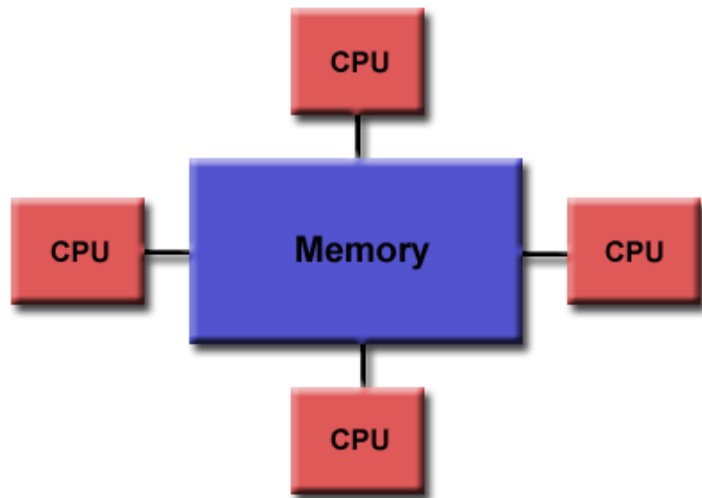


Example: Networks connect multiple standalone computers (nodes) to make larger parallel computer clusters.

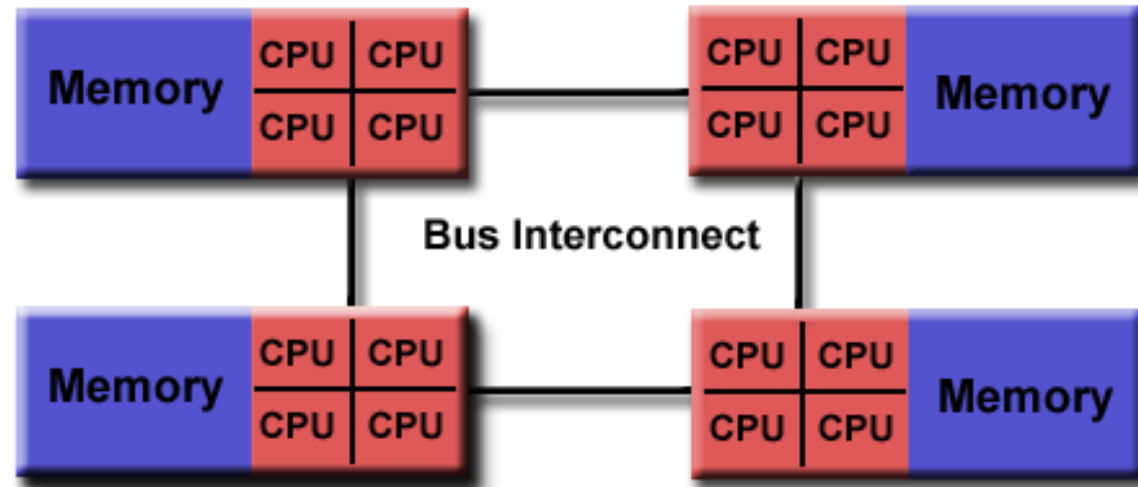
# Parallel Computer Memory Architectures

Shared Memory – sharing the same address space

*Symmetric Multiprocessor (SMP)* machines



Shared Memory (UMA)



Shared Memory (NUMA)

# Parallel Computer Memory Architectures

## Shared Memory

- Advantages

- Global address space provides a user-friendly programming perspective to memory
- Data sharing between tasks is fast
- NUMA - One SMP can directly access memory of another SMP

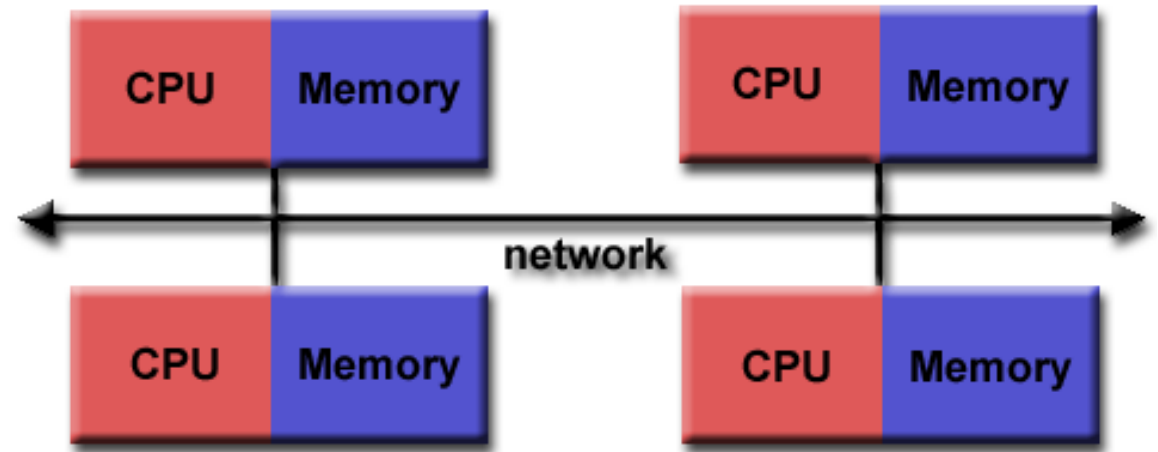
- Disadvantages

- Lack of scalability between memory and CPUs.
- Programmer responsibility for synchronization constructs that ensure "correct" access of global memory.
- NUMA - unequal access time to all memories, Memory access across link is slower

# Parallel Computer Memory Architectures

## Distributed Memory

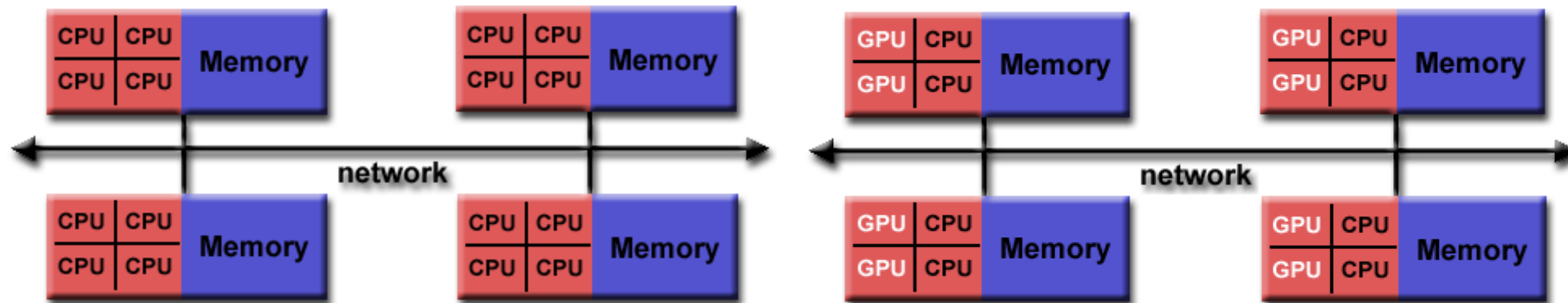
- A communication network to connect inter-processor memory.
- Processors have their own local memory. No concept of global address space across all processors.
- Changes it makes to its local memory have no effect on the memory of other processors.
- Task of the programmer to explicitly define how and when data is communicated. Synchronization between tasks is likewise the programmer's responsibility.
- The network "fabric" used for data transfer varies widely



# Parallel Computer Memory Architectures

## Hybrid Distributed -Shared Memory

- The shared memory component can be a shared memory machine and/or graphics processing units (GPU).
- network communications are required to move data from one machine to another.
- Increased programmer complexity/effort



# Parallel Computing Terminology

- **Supercomputing / High Performance Computing (HPC)** : Using the 'class' of fastest and largest computers to solve large problems.
- **Node** : a standalone "computer in a box". Usually comprised of multiple CPUs/processors/cores, memory, network interfaces, etc. Nodes are networked together to comprise a supercomputer.
- **CPU / Processor / Core** : It Depends..... (A Node has multiple Cores or Processors)
- **Rank** – Identifying number of a process

# Limits and Costs of Parallel Programming

Parallel programs contain

- Serial Section
- Parallel Section
- Observed speedup of a code which has been parallelized, defined as:

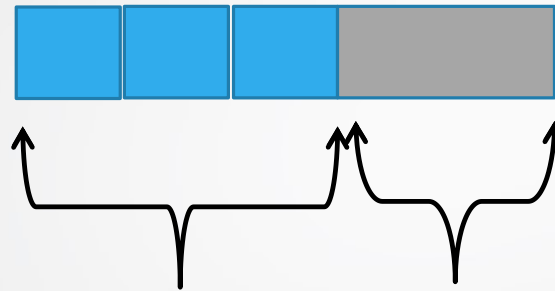
$$\frac{\text{wall-clock time of serial execution}}{\text{wall-clock time of parallel execution}}$$



# Amdhal's Law

Speedup is limited by the non-parallelizable/ serial portion of the work.

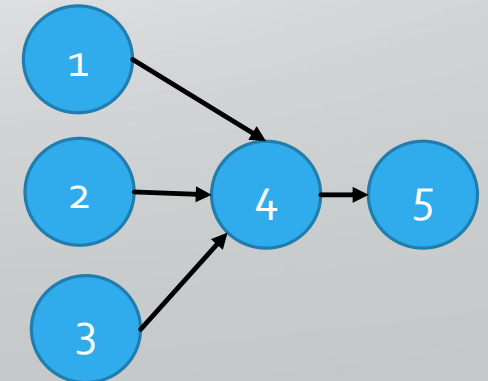
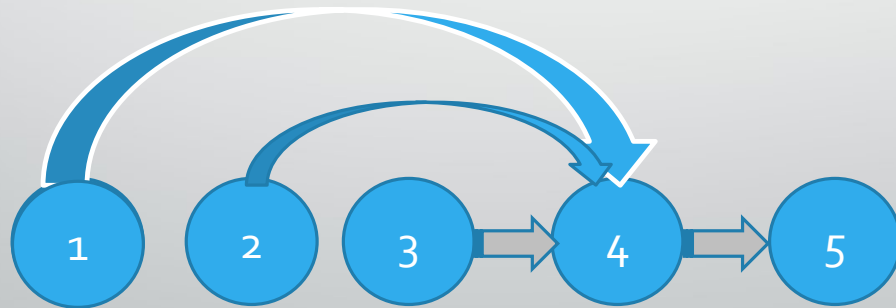
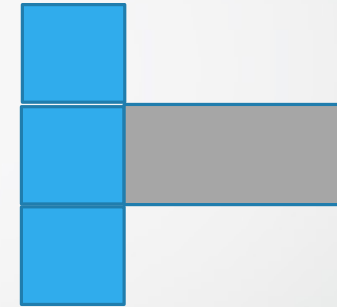
Time = 5 units



Parallelizable

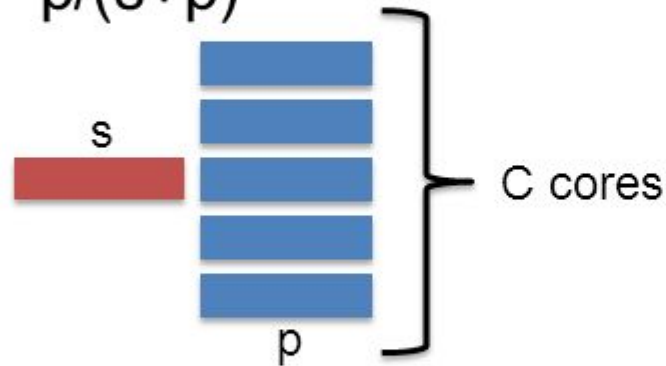
Serial

Time = 3 units



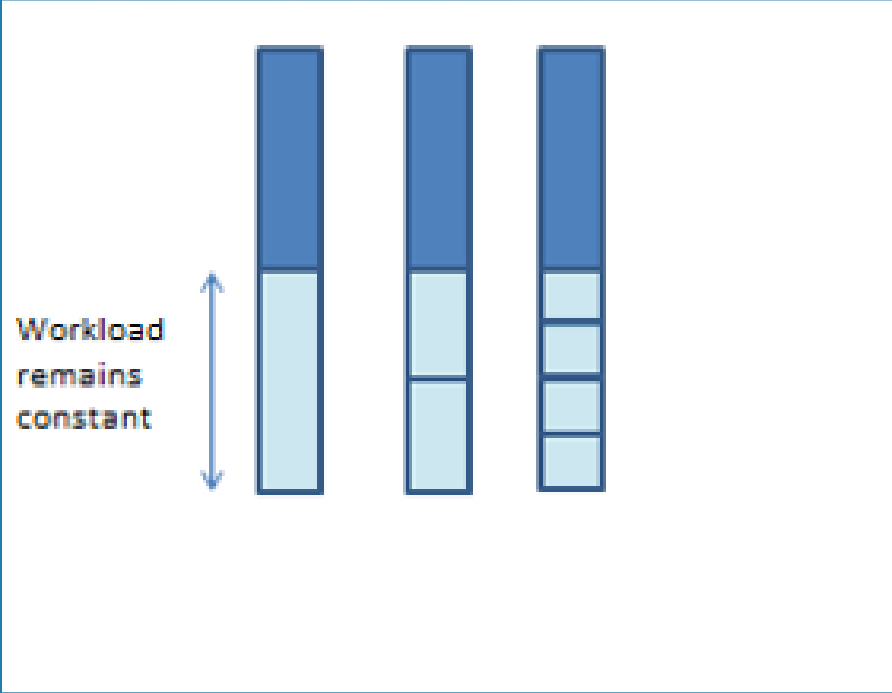
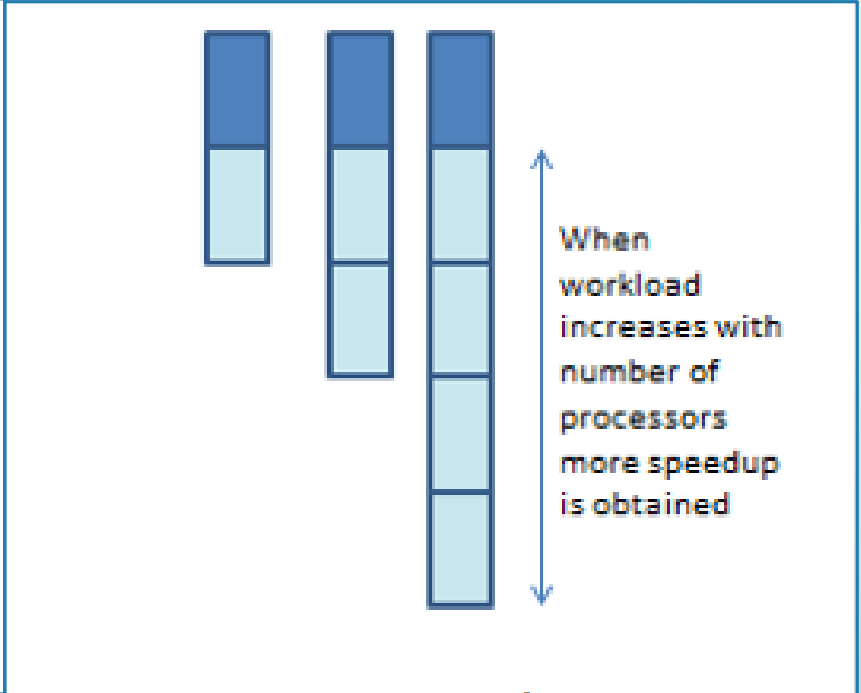
# Gustafson's Law

- › As more cores are integrated, the workloads are also growing!
- › Let  $s$  be the serial time of a program and  $p$  the time that can be done in parallel
- › Let  $f = p/(s+p)$



$$\text{Speedup} = \frac{s + pC}{s + p} = 1 - f + fC = 1 + f(C - 1)$$

# Amdhal's Law & Gustafson's Law

 <p>The diagram shows three vertical bars representing the workload on 1, 2, and 3 processors respectively. Each bar is divided into a light blue bottom section and a dark blue top section. The total height of each bar is constant, representing a fixed workload. As the number of processors increases, the light blue section (representing parallelizable work) is divided among more processors, while the dark blue section (representing sequential work) remains the same size. A vertical double-headed arrow on the left is labeled "Workload remains constant".</p>	 <p>The diagram shows three vertical bars representing the workload on 1, 2, and 3 processors respectively. The total height of each bar increases as the number of processors increases, representing an increasing workload. Each bar is divided into a light blue bottom section and a dark blue top section. The light blue section is divided among the processors, and the dark blue section also increases in size. A vertical double-headed arrow on the right is labeled "When workload increases with number of processors more speedup is obtained".</p>
<p>Amdhal's Law / Strong Scaling</p>	<p>Gustafson's Law / Weak Scaling</p>
<p>How quickly can we complete analysis on a particular data set by increasing Processor count?</p>	<p>Can we analyze more data in approx. same amount of time by increasing Processor count?</p>

# Designing Parallel Programs

- Can the problem be parallelized?
  - Calculation of the Fibonacci series (0,1,1,2,3,5,8,13,21,...) by use of the formula:  $F(n) = F(n-1) + F(n-2)$
- Identify the program's **hotspots** (*real work*)
- Identify **bottlenecks** in the program
- Identify Data Dependencies and Task Dependencies (inhibitors to parallelism )
- Investigate other algorithms if possible & take advantage of optimized third party parallel software.
- third party parallel software and highly optimized math libraries available

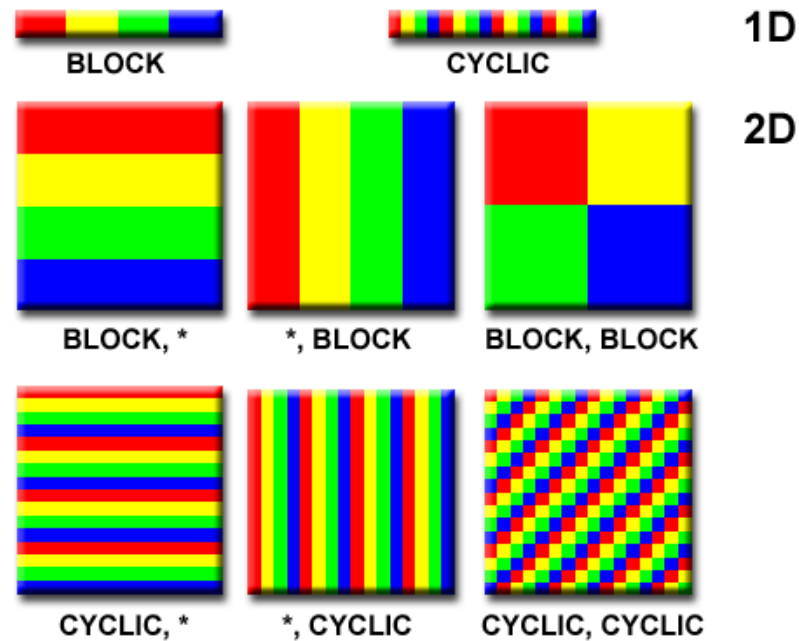
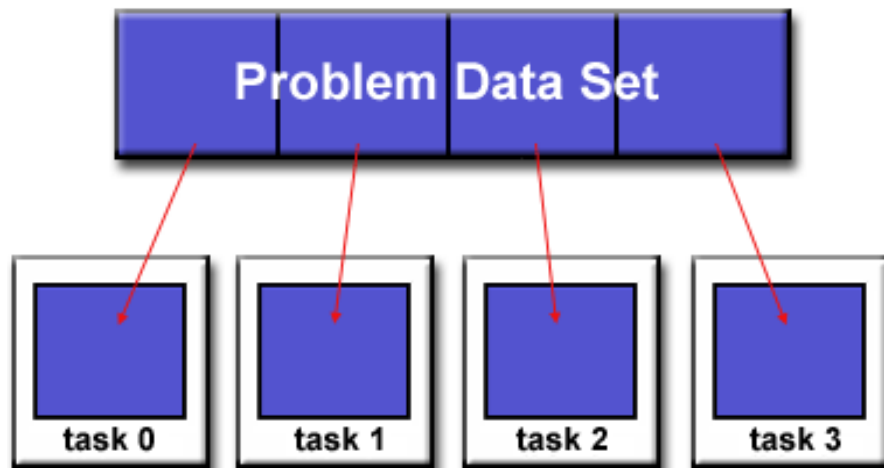
# Designing Parallel Programs

## Partitioning

- ***domain decomposition*** and ***functional decomposition***

### 1. Domain/Data Decomposition

Data associated with the problem is decomposed



# Data Decomposition

For problems that operate on large amounts of data Data is divided up between CPUs : Each CPU has its own chunk of dataset to operate upon and then the results are collated.

Which data should we partition?

Input Data

Output Data

Intermediate Data

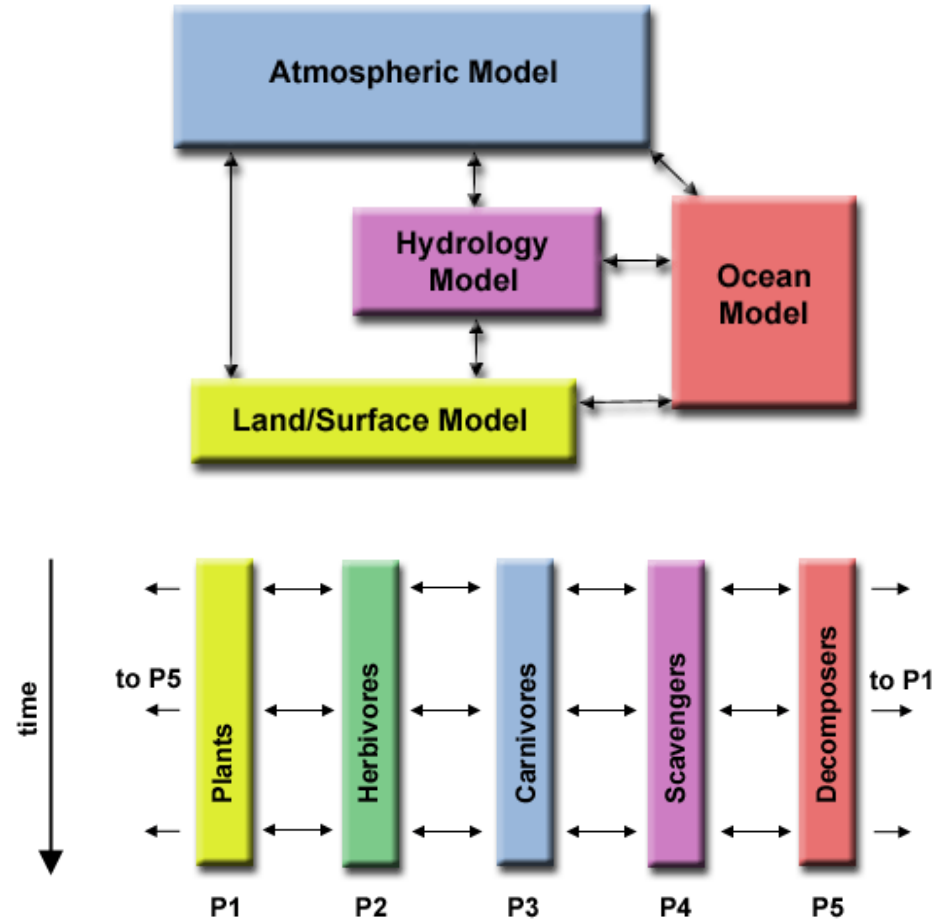
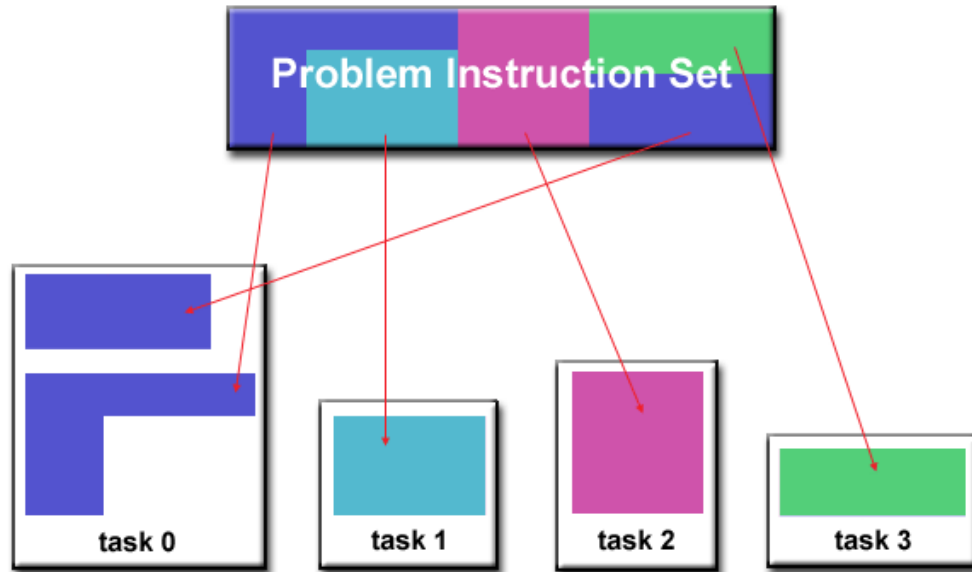
Ensure Load Balancing : Equal sized tasks not necessarily equal size data sets.

- Static Load Balancing
- Dynamic Load Balancing

# Designing Parallel Programs

- **Functional/ Task Decomposition**

- The problem is decomposed according to the work that must be done. Each task then performs a portion of the overall work



# Data Dependencies

- The order of statement execution affects the results of the program.
- Multiple use of the same location(s) in storage by different tasks.

```
DO J = MYSTART, MYEND  
A(J) = A(J-1) * 2.0  
END DO
```

Loop carried dependence

Task 1	Task 2
-----	-----
<b>X = 2</b>	<b>X = 4</b>
....	....
....	....
<b>Y = X**2</b>	<b>Y = X**3</b>

Loop independent data dependency



# Data Dependencies

```
DO J = MYSTART, MYEND  
A(J) = A(J-1) * 2.0  
END DO
```

If Task 2 has A(J) and task 1 has A(J-1)

- *Distributed memory architecture* - task 2 must obtain the value of A(J-1) from task 1 after task 1 finishes its computation
- *Shared memory architecture* - task 2 must read A(J-1) after task 1 updates it

Task 1	Task 2
-----	-----
<b>X = 2</b>	<b>X = 4</b>
....	....
<b>Y = X**2</b>	<b>Y = X**3</b>

(Race Condition) The value of Y is dependent on:

*Distributed memory architecture* - if or when the value of X is communicated between the tasks.

*Shared memory architecture* - which task last stores the value of X

# Handling Dependencies

- Distributed memory architectures - communicate required data at synchronization points
- Shared memory architectures -synchronize read/write operations between tasks.
  - Data Dependencies:- Mutual Exclusion, Locks & Critical Sections
- Task Dependencies:- Explicit or Implicit Synchronization points called Barriers

# Load Distribution

GOAL : Assigning the tasks/ processes to Processors while Minimizing Parallel Processing Overheads

- Maximize data locality
- Minimize volume of data-exchange
- Minimize frequency of interactions
- Minimize contention and hot spots
- Overlap computation with interactions
- Selective data and computation replication

THANK YOU