High Performance Computing

Introduction to Parallel Computing

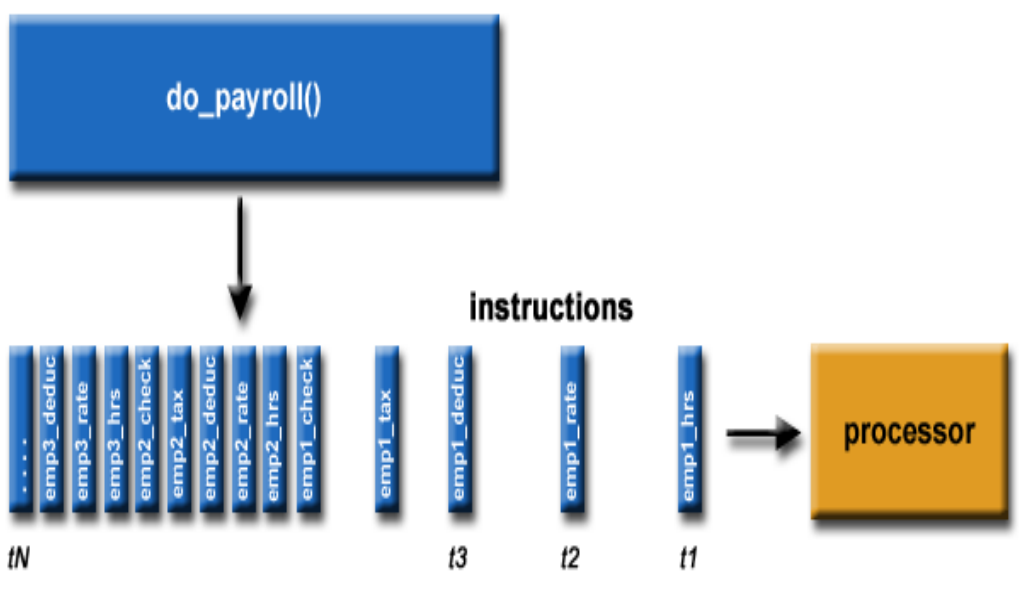
Acknowledgements

Content of the following presentation is borrowed from

The Lawrence Livermore National Laboratory

https://hpc.llnl.gov/training/tutorials

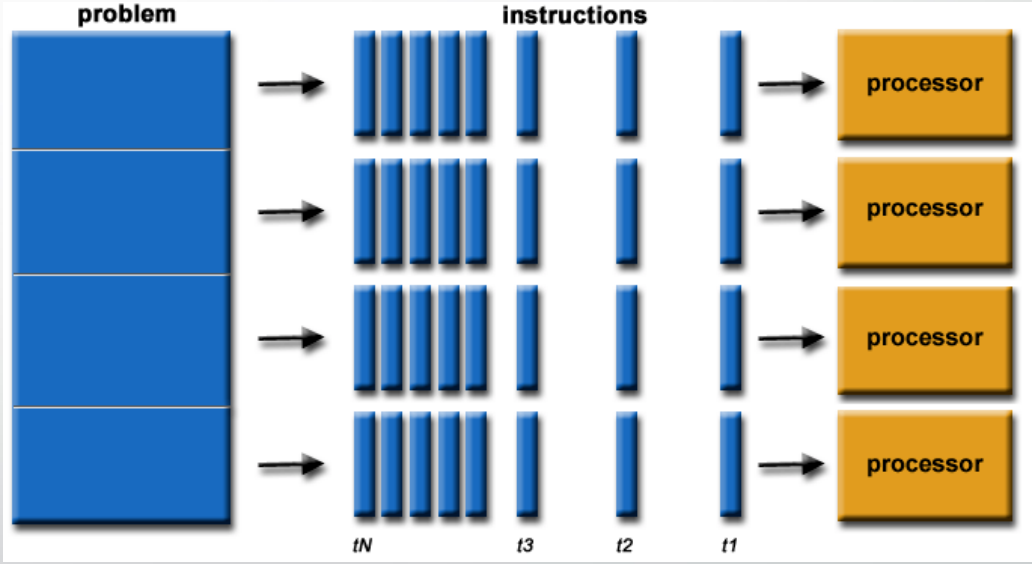
Serial Computing



- 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

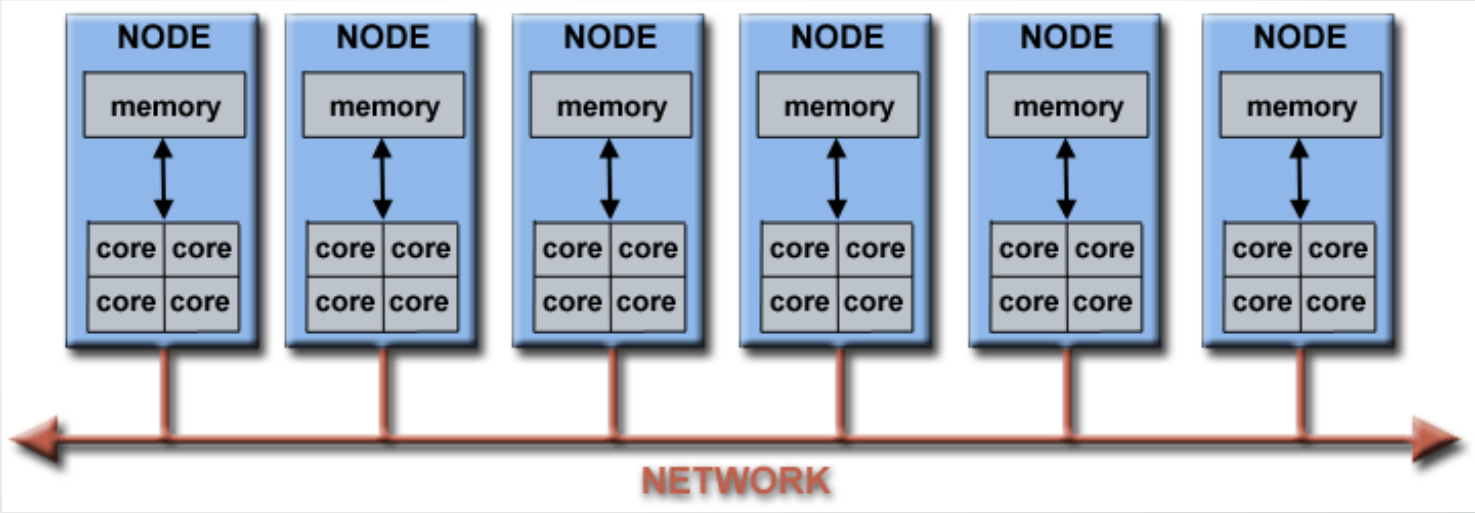
- Simultaneous use of multiple compute resources to solve a computational problem.
- Run on multiple CPUs
- Problem is decomposed into multiple parts that can be solved concurrently.
- Each part is decomposed into a set of instructions.
- Instructions are executed simultaneously on different CPUs



Parallel Computer Architecture

Compute Resources

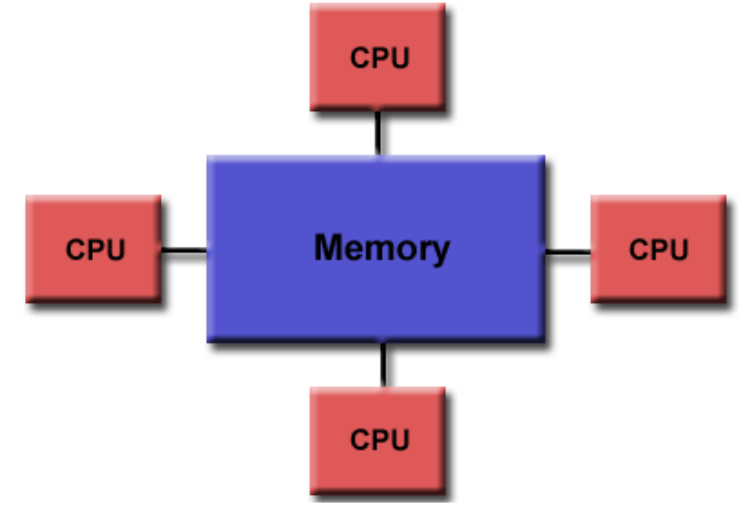
- Single Computer with multiple processors.
- Arbitrary Number of Computers connected by a network.
- Combination of both

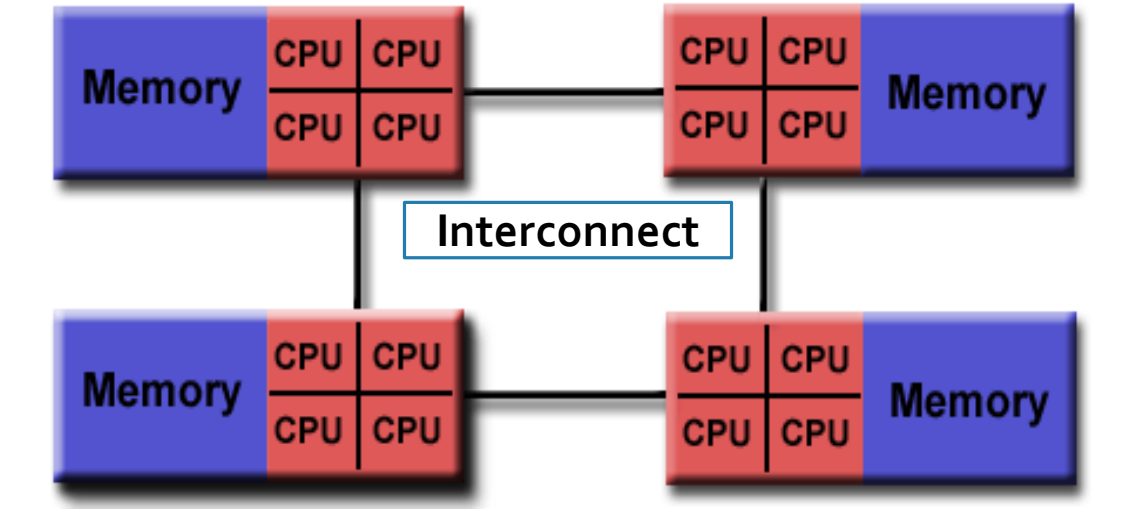


Example: Networks connect multiple stand-alone computers (nodes) to make larger parallel computer clusters.

Parallel Computer Memory Architectures

Shared Memory





Sharing the same address space

Parallel Computer Memory Architectures
Shared Memory

Advantages:

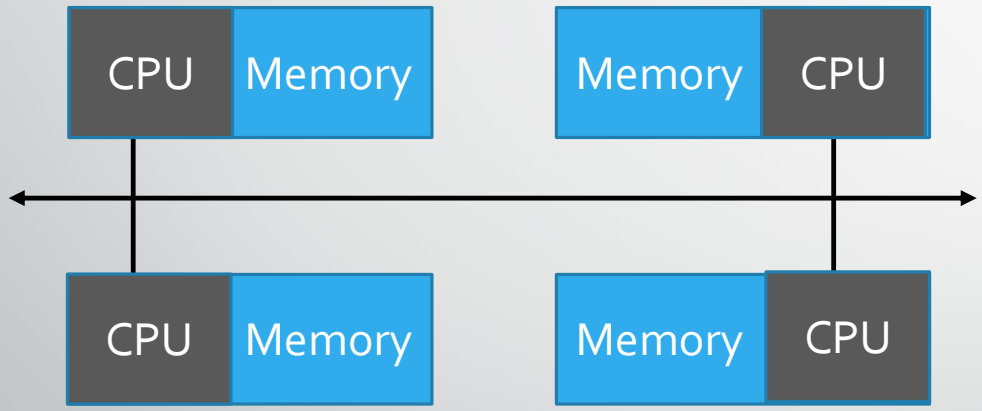
- Global address space provides a user-friendly programming perspective to memory
- Data sharing between tasks is both fast and uniform due to the proximity of memory to CPUs

Disadvantages:

- Lack of scalability between memory and CPUs.
- Programmer responsibility for synchronization constructs that ensure "correct" access of global memory.

Parallel Computer Memory Architectures

Distributed Memory



A communication network to connect inter-processor memory.

Processors have their own local memory. Memory addresses in one processor do not map to another processor, so there is 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

Distributed Memory

Advantages:

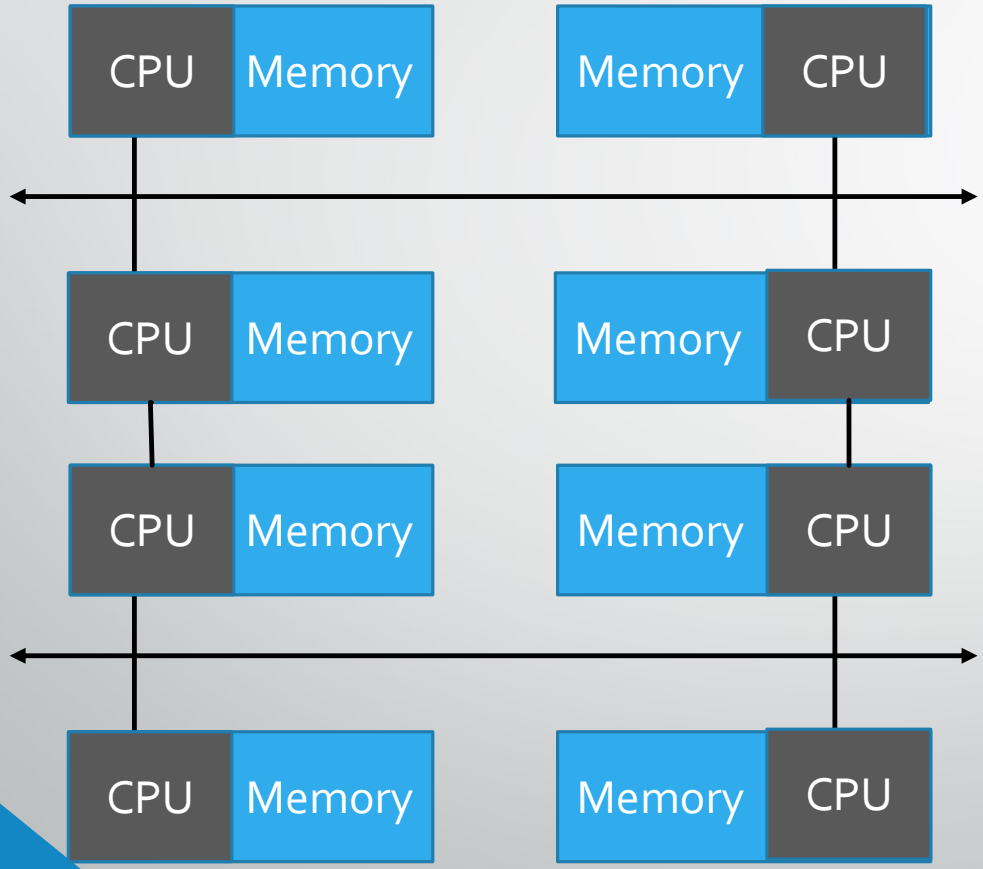
- Scalable Memory: is scalable with the number of processors.
- Each processor can rapidly access its own memory.
- Cost effective: can use commodity, off-the-shelf processors and networking.

Disadvantages:

- Programmer is responsible for details associated with data communication between processors.
- It may be difficult to map existing data structures, based on global memory, to this memory organization.
- Non-uniform memory access times - data residing on a remote node takes longer to access than node local data.

Parallel Computer Memory Architectures

Hybrid Memory



The largest and fastest computers in the world today employ both shared and distributed memory architectures

The shared memory component can be a shared memory machine and/or graphics processing units (GPU).

The distributed memory component is the networking of multiple shared memory/GPU machines, which know only about their own memory - not the memory on another machine.

Network communications are required to move data from one machine to another.

Trends indicate that this type of memory architecture will prevail and increase at the high end of computing for the foreseeable future,

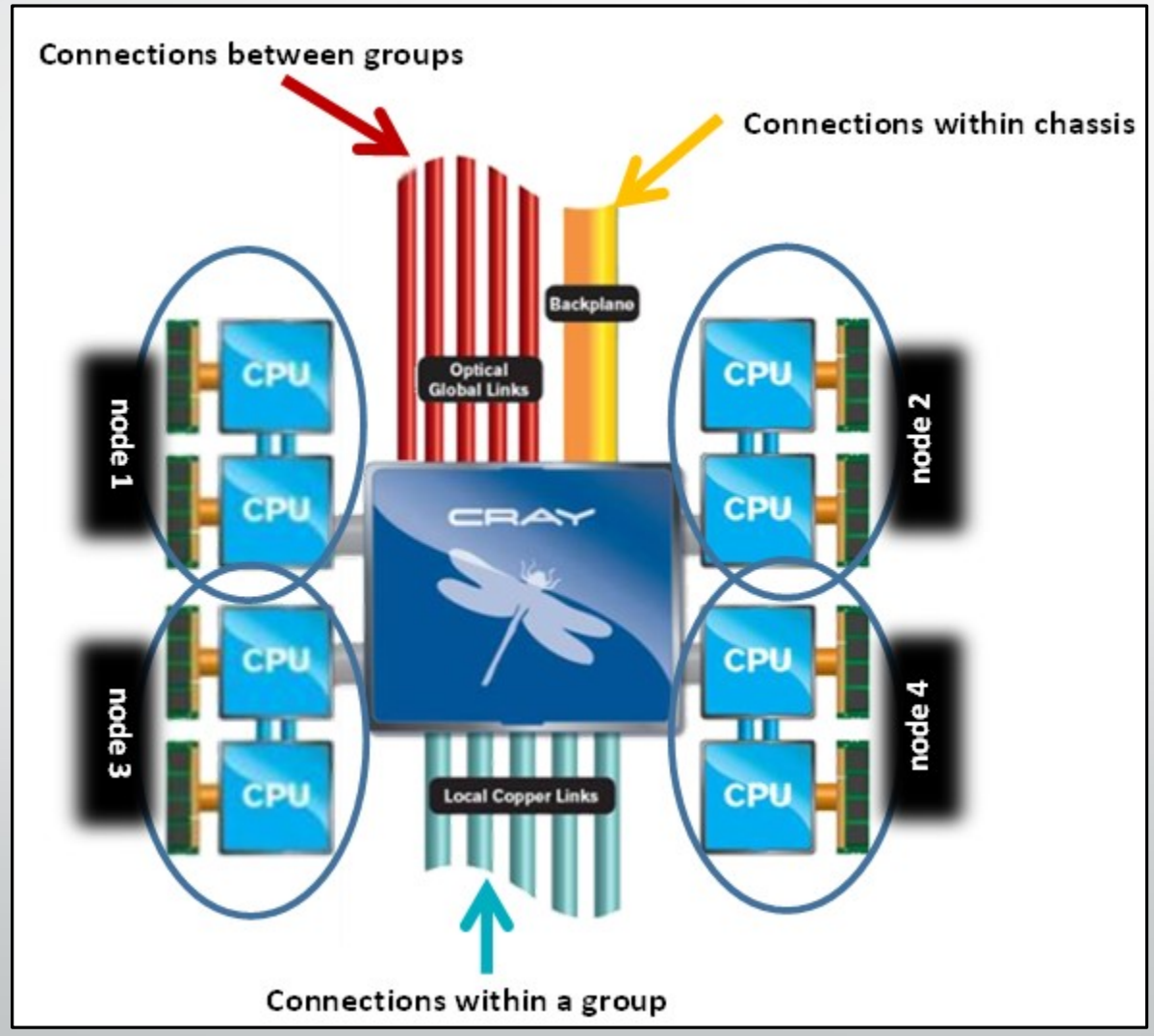
The Parallel Computing Terminology

Supercomputing / High Performance Computing (HPC) : Using the world's 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)

Logical View of a Supercomputer



Limits and Costs of Parallel Programming

Parallel programs contain

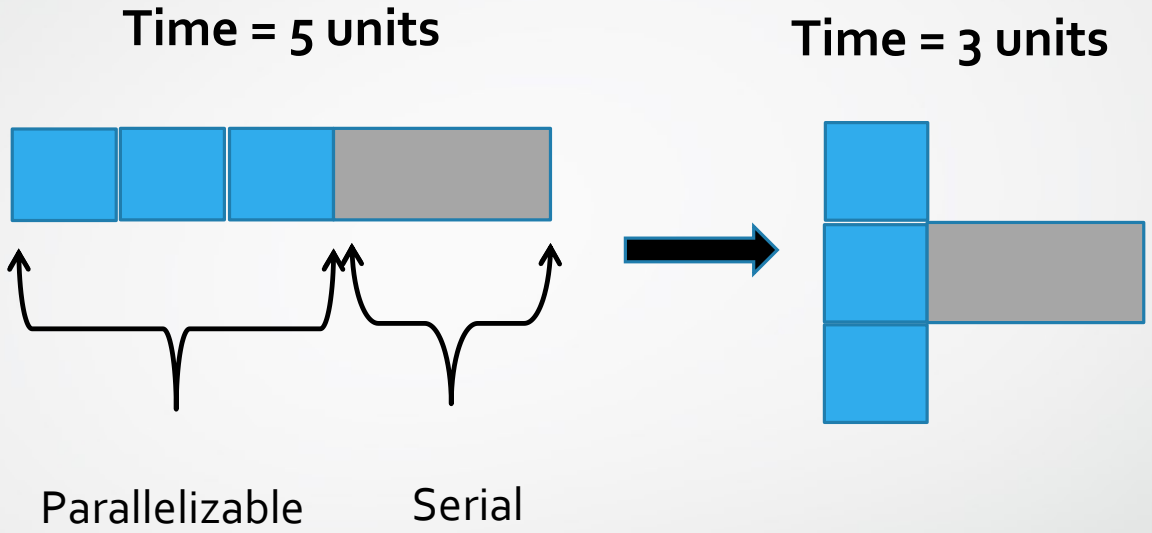
- Serial Section
- Parallel Section

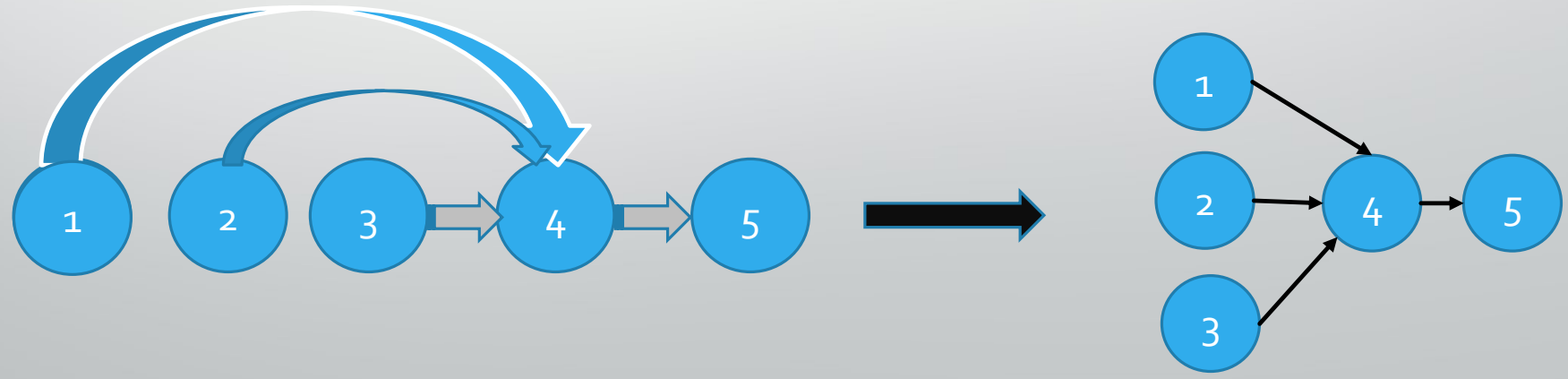
Observed speedup of a code which has been parallelized, defined as:

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

Amdhal's Law

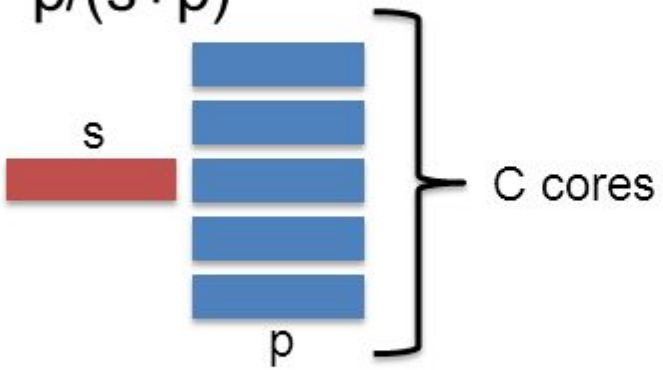
Speedup is limited by the non-parallelizable/ serial portion of the work.





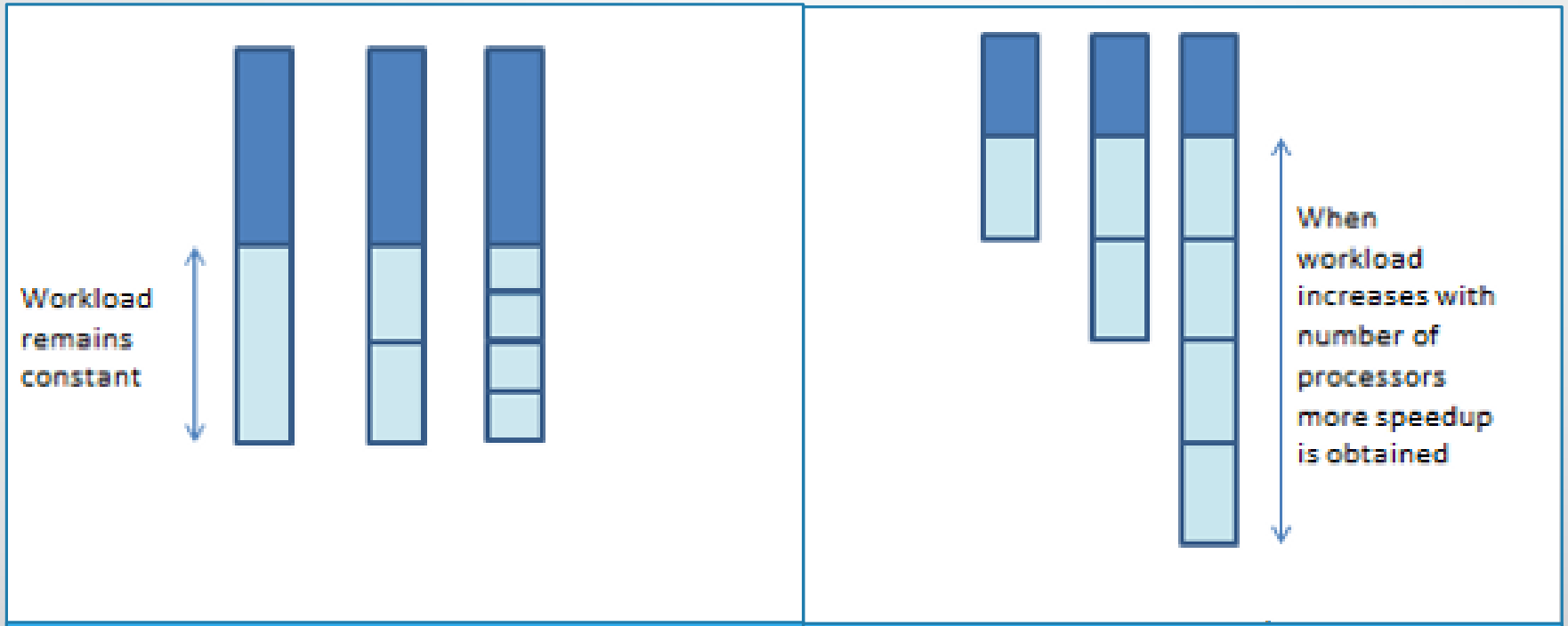
Gustafson's Law

- 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)



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

Amdhal's Law & Gustafson's Law

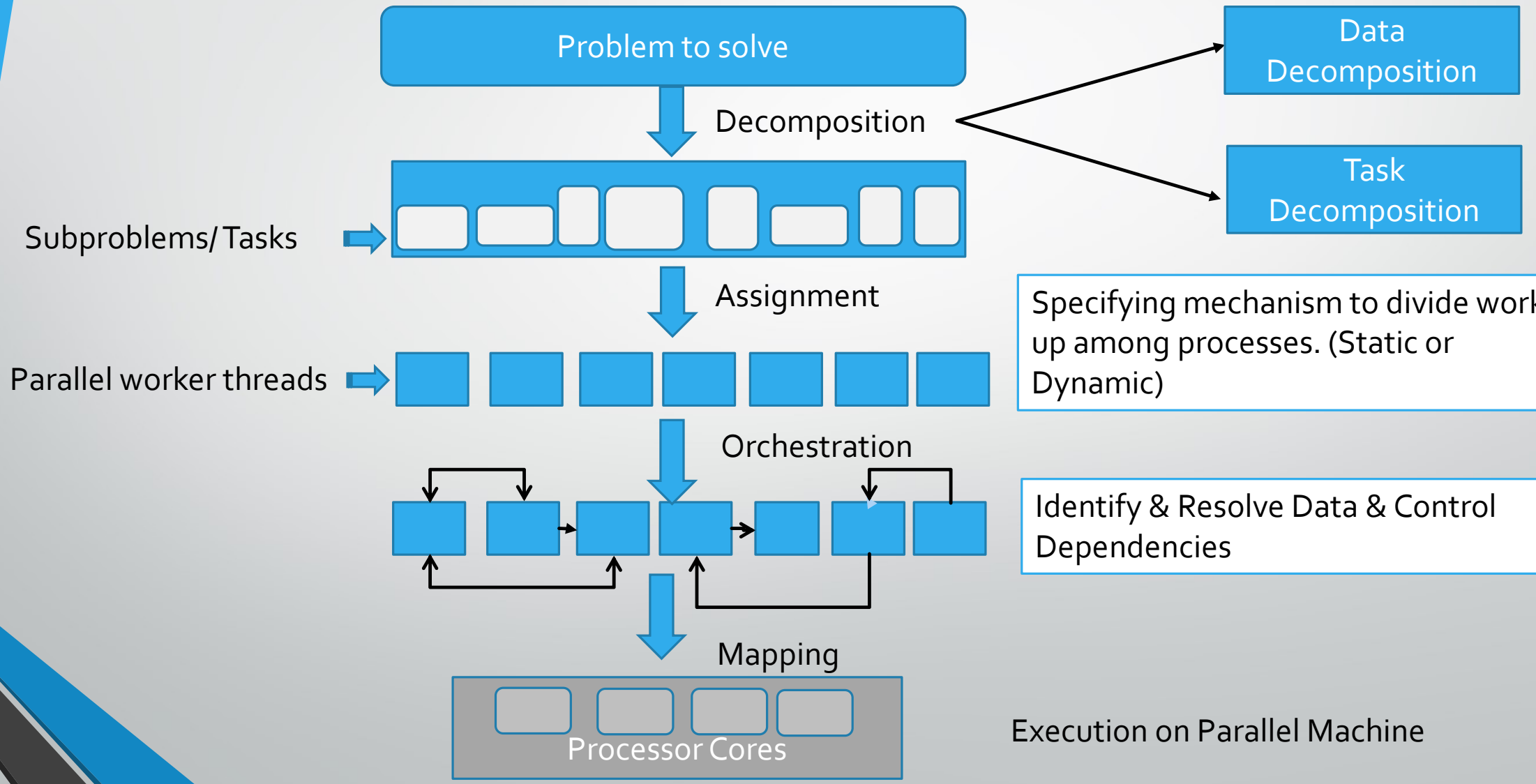


Amdhal's Law / Strong Scaling	Gustafson's Law / Weak Scaling
How quickly can we complete analysis on a particular data set by increasing Processor count?	Can we analyze more data in approx. same amount of time by increasing Processor count?

Moving from Serial to Parallel Code

- 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***
- 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.

Moving from Serial to Parallel Code



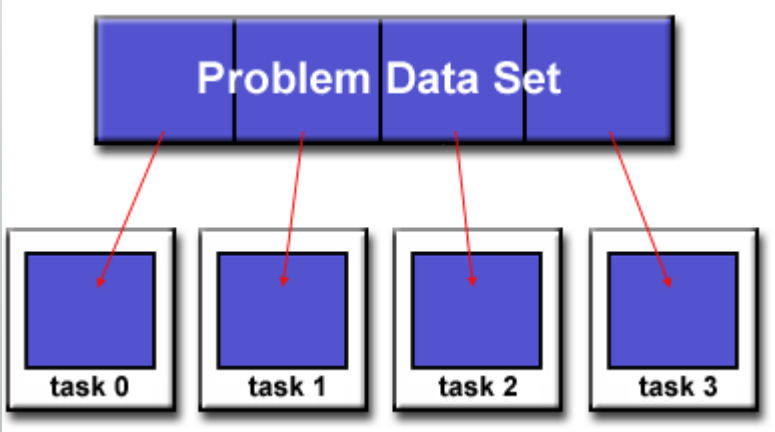
Moving from Serial to Parallel Code : Decomposition

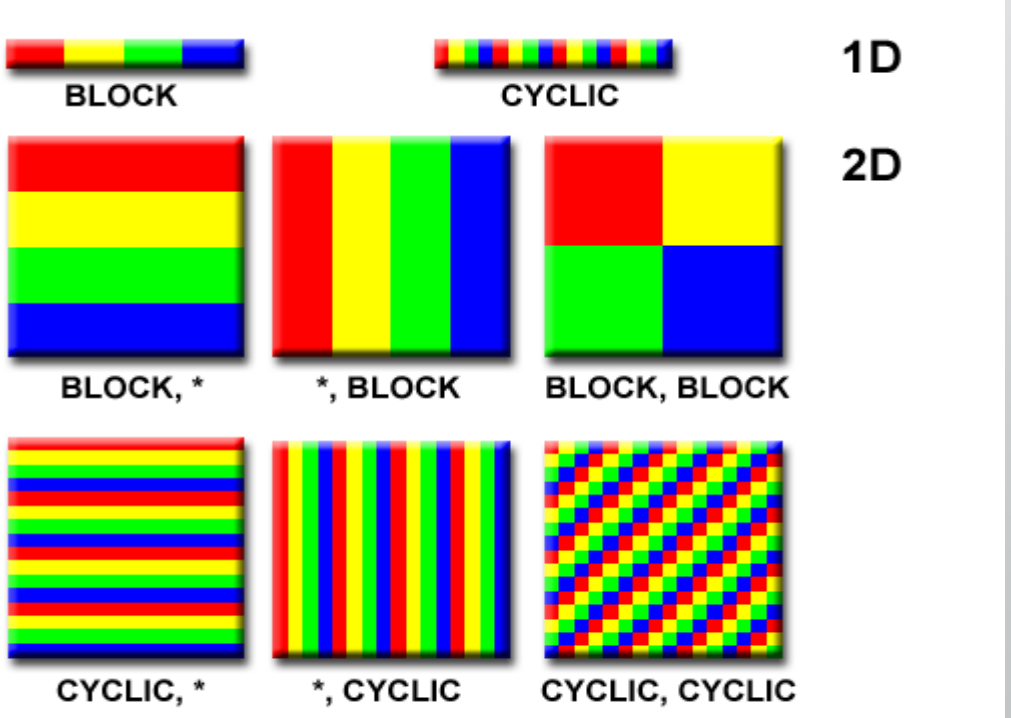
Decomposition

Divide computation into smaller parts(tasks) that can be executed concurrently
There are two basic ways to partition computational work among parallel tasks

1. Domain/Data Decomposition

Data associated with the problem
is decomposed





More on 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

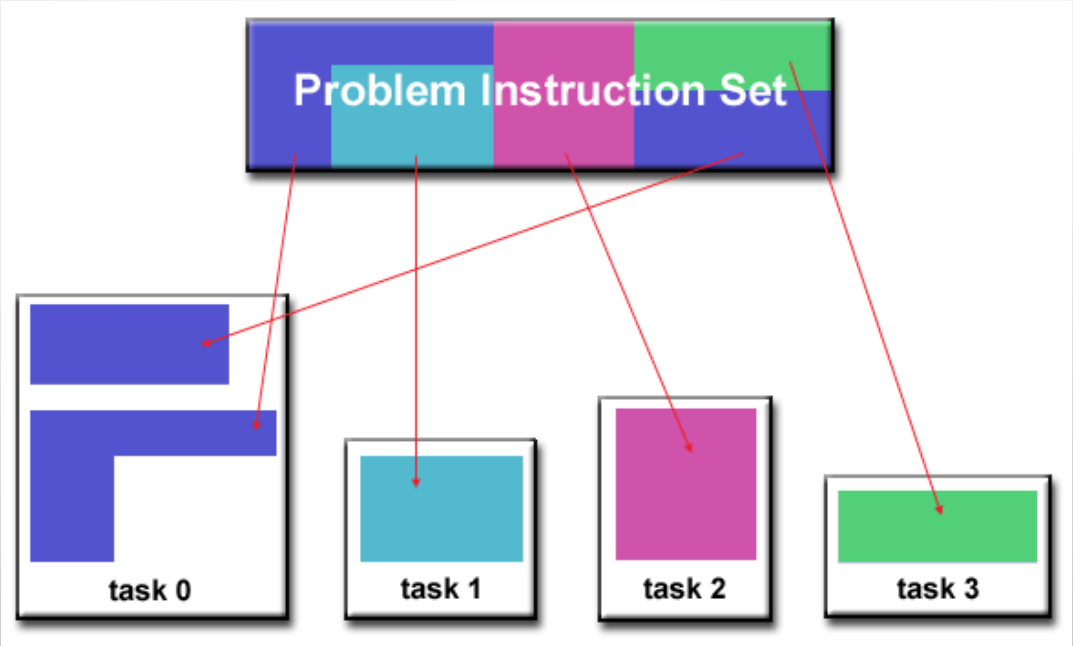
Ensure Load Balancing : Equal sized tasks not necessarily equal size data sets.

- Static Load Balancing
- Dynamic Load Balancing

Moving from Serial to Parallel Code: Decomposition

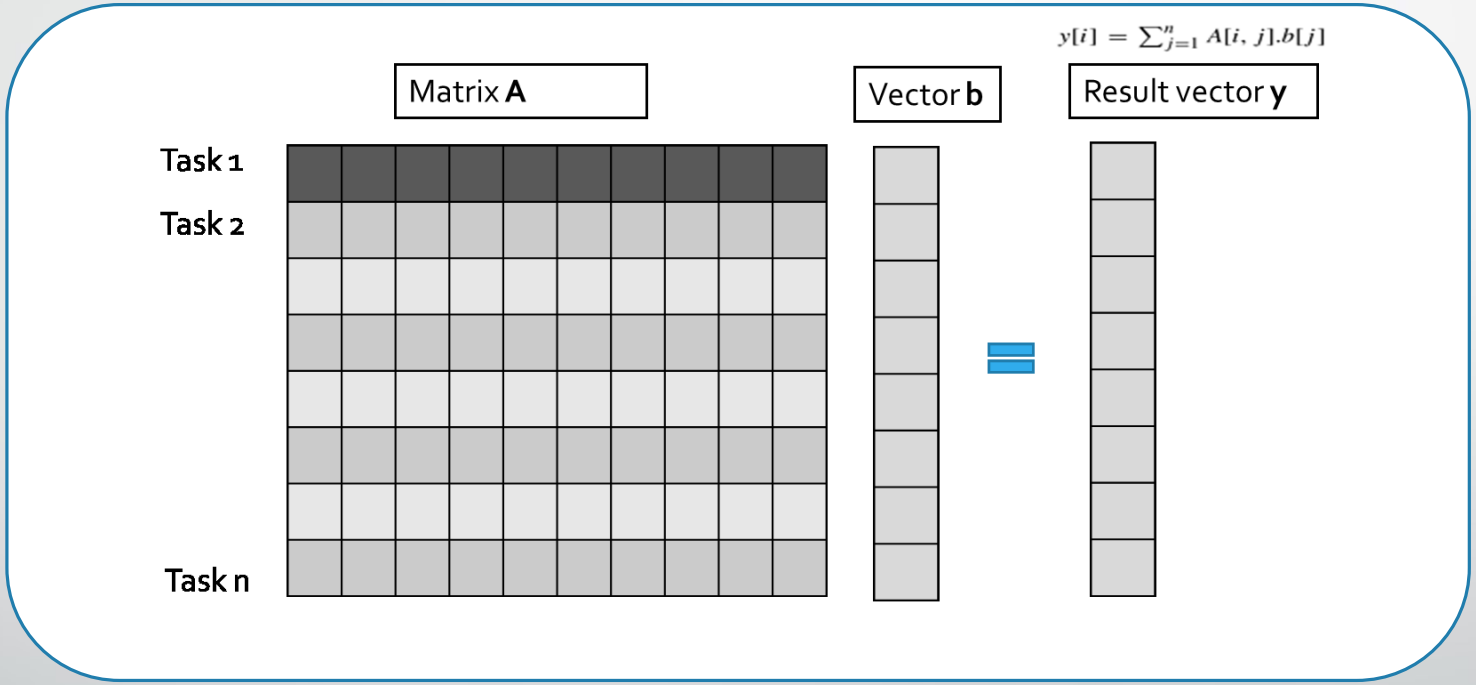
2. Functional/ Task Decomposition

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



Decomposition : Example

Dense Matrix-Vector Multiplication



Computing y[i] only use ith row of A and b
Task = > computing y[i]
Task size is uniform
No dependence between tasks
All tasks need b

Decomposition : What to look for

- Does the partition define (at least an order of magnitude) more tasks than there are processors in your target computer? (Flexibility)
- Does the partition avoid redundant computation and storage requirements? (Scalability)
- Are tasks of comparable size? (Load Balancing)
- Does the number of tasks scale with problem size? Increase in problem size should increase the number of tasks rather than the size of individual tasks.
- Identify alternative partitions? You can maximize flexibility in subsequent design stages by considering alternatives now.
- Investigate both domain and functional decompositions.

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          Task2
------          ------
X = 2           X = 4
. . . .
.............
Y = X**2        Y = X**3
```

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          Task2
------          ------
X = 2           X = 4
. . . .         ..........
Y = X**2        Y = X**3
```

(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

Mapping

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

HPC Solution Stack