Introduction to HPC and Parallel Programming

Prachi Pandey

System Software Development Group

C-DAC, Bengaluru

prachip@cdac.in

Agenda

- Parallel Computing
  - Need for Parallel computing
  - Parallel architectures
  - Cluster computing
  - Memory architectures
  - Parallel Computing classification
- High Performance Computing
  - Applications
  - Top supercomputers
- Parallel Programming
  - Decomposition/partitioning
  - Dependency analysis
  - Granularity
  - Amdahl's law
  - Parallel programming paradigms
  - Parallel programming approach

Parallel Computing

Serial Computing

If 1 Man takes 10 hours to complete the job; then

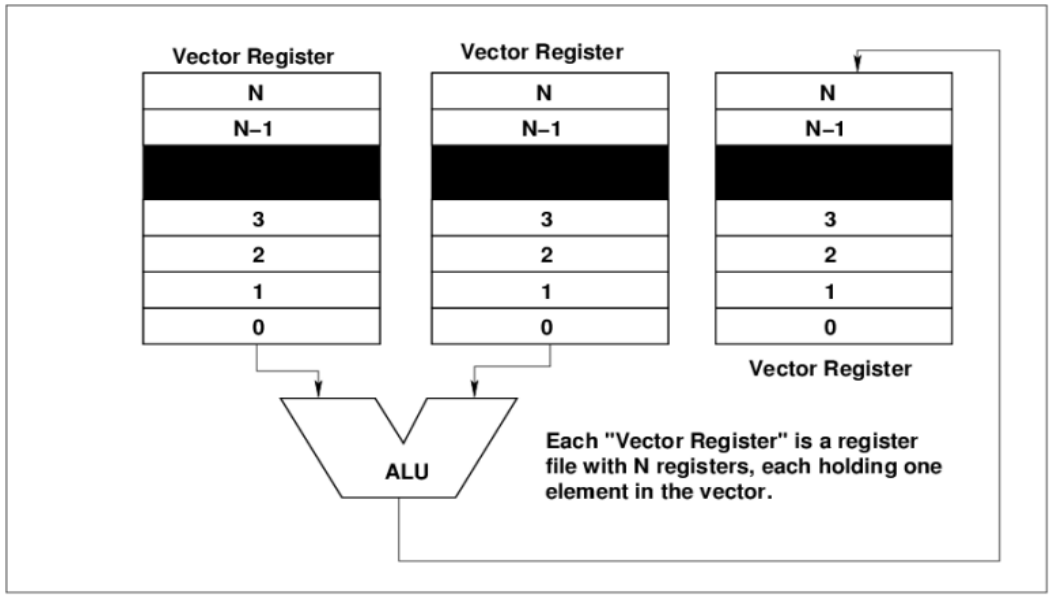
How long will it take if 10 Men work together

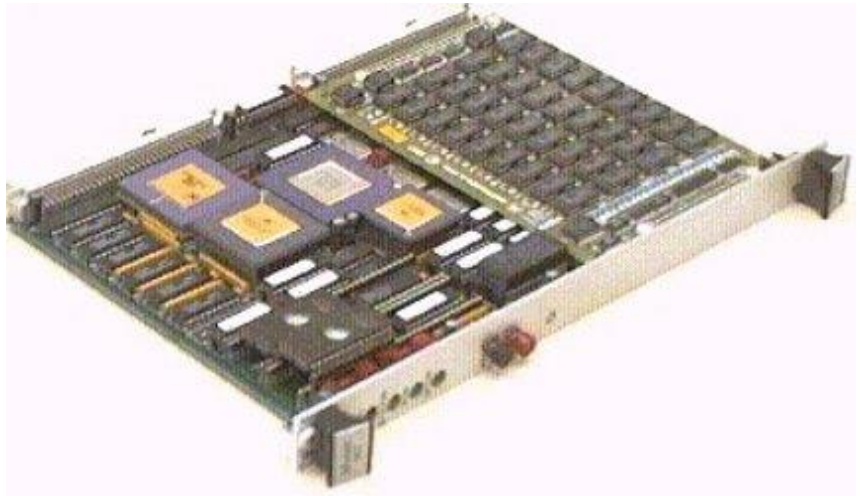
Parallel Computing!

Parallel Architectures

Vector Processors

Able to run mathematical operations on multiple data elements simultaneously

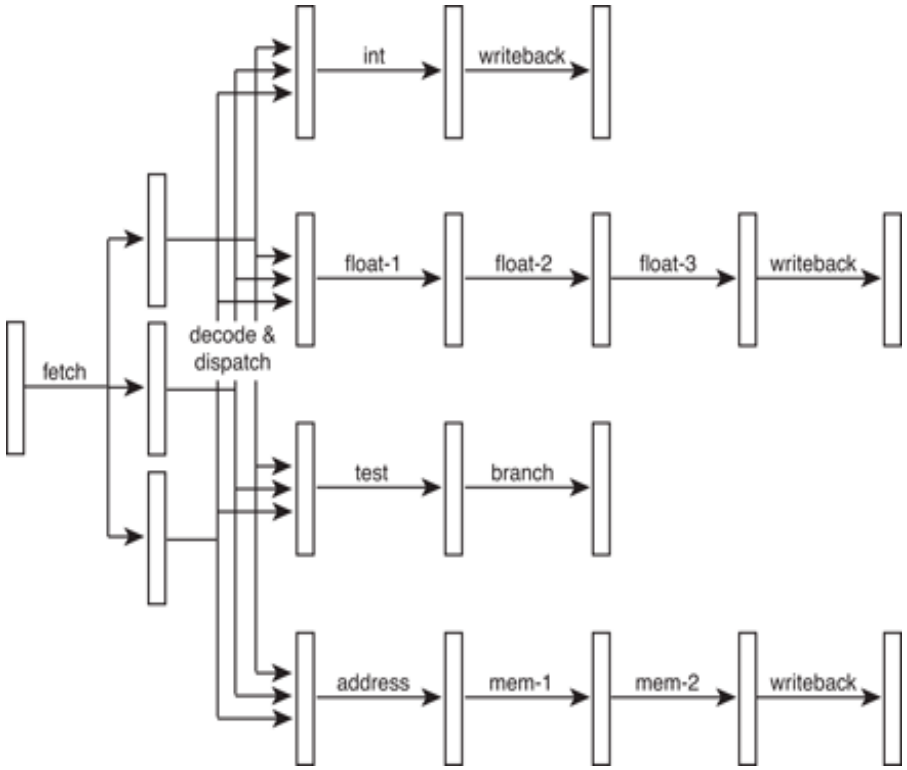


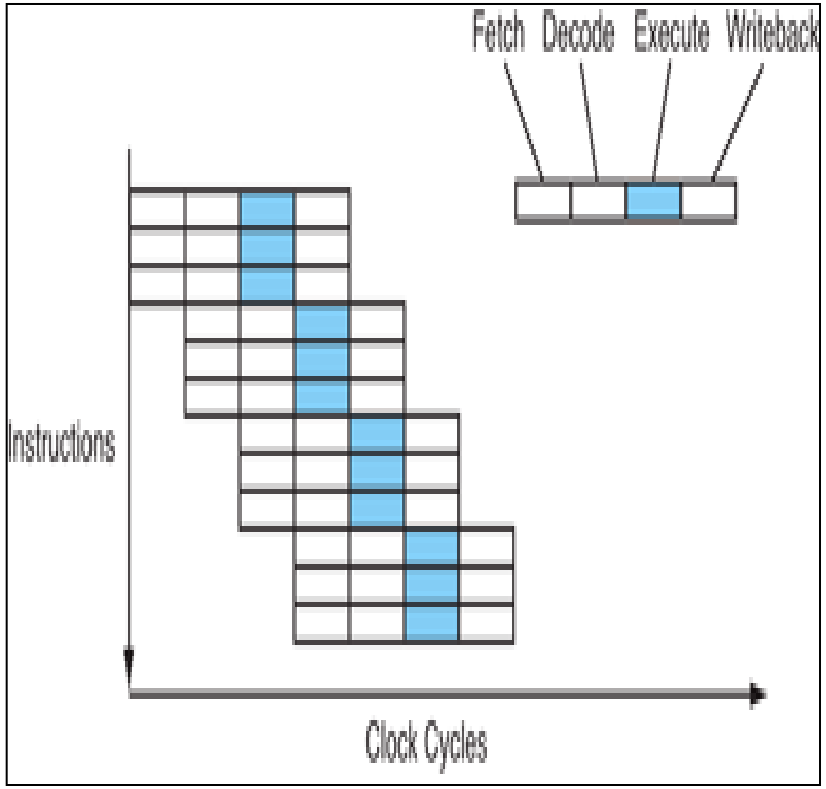


http://www.ausairpower.net/OSR-0600.html

Superscalar Processors

Instruction level parallelism with a processor



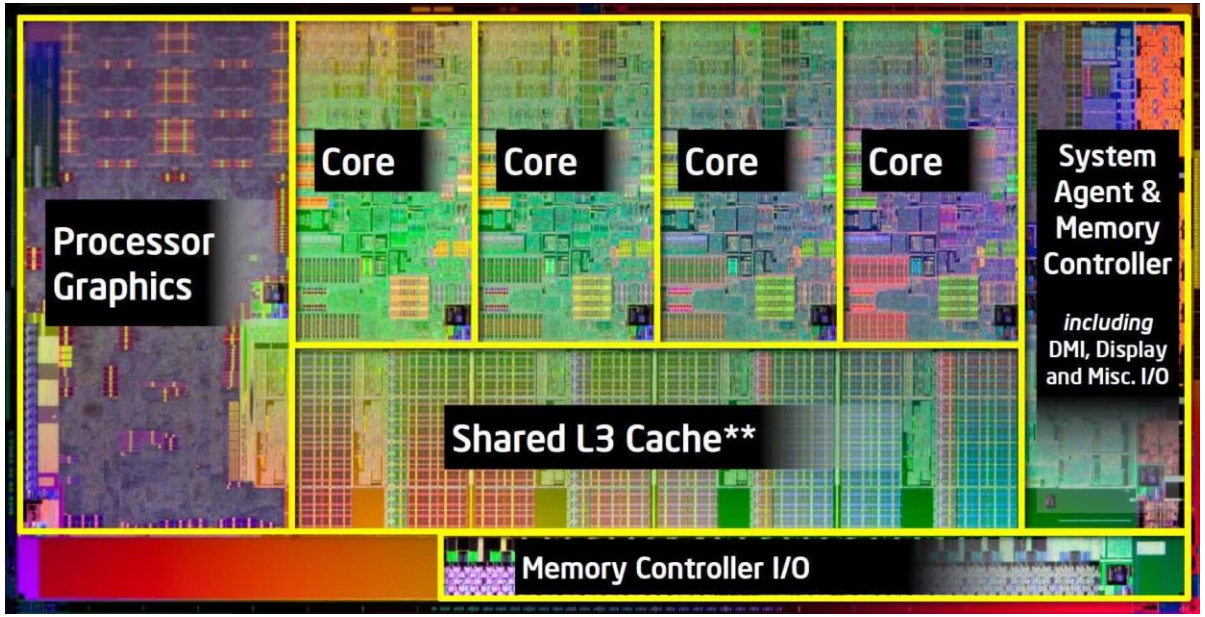


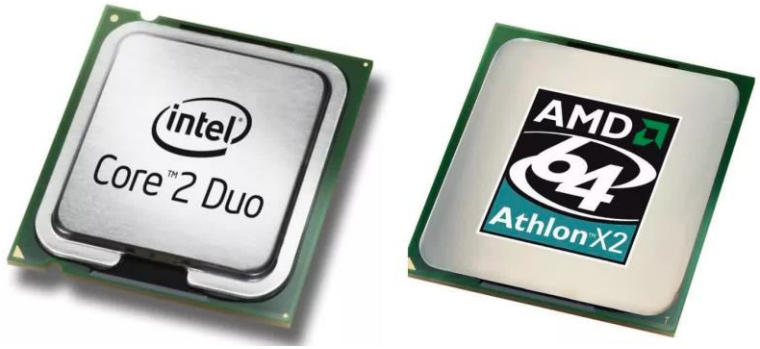
Eg: Intel i960CA, AMD-29000series

http://www.lighterra.com/papers/modernmicroprocessors/

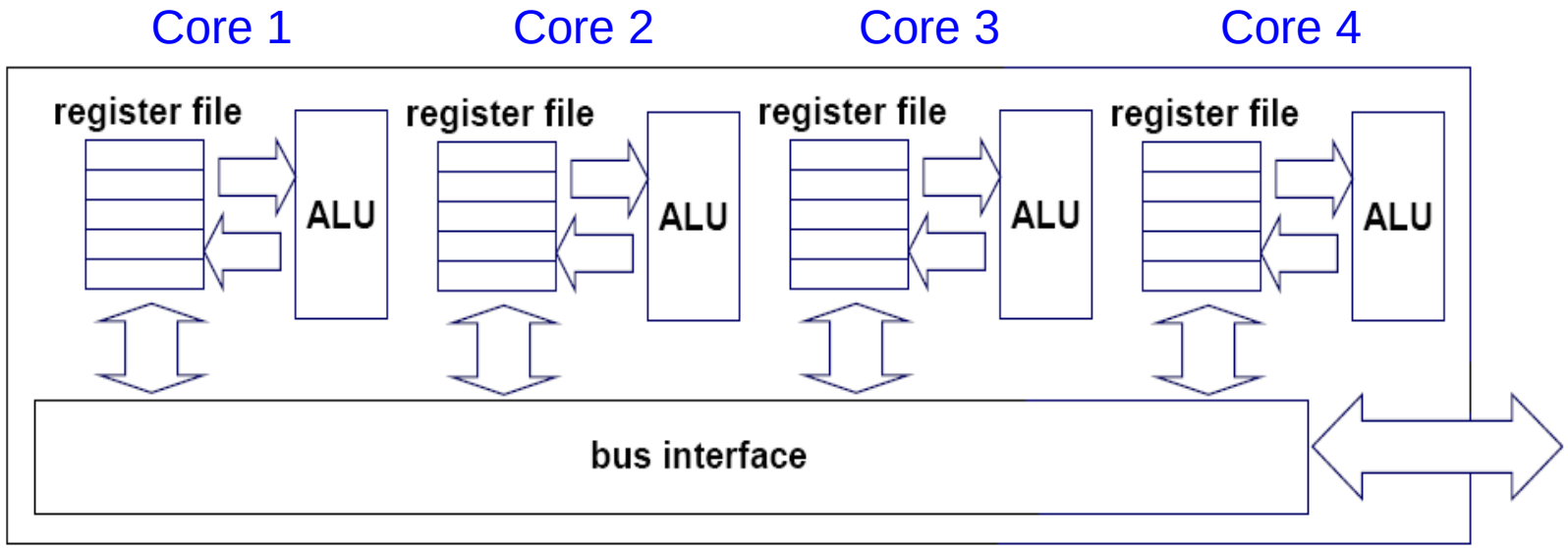
Parallel architectures

MultiCore Chips





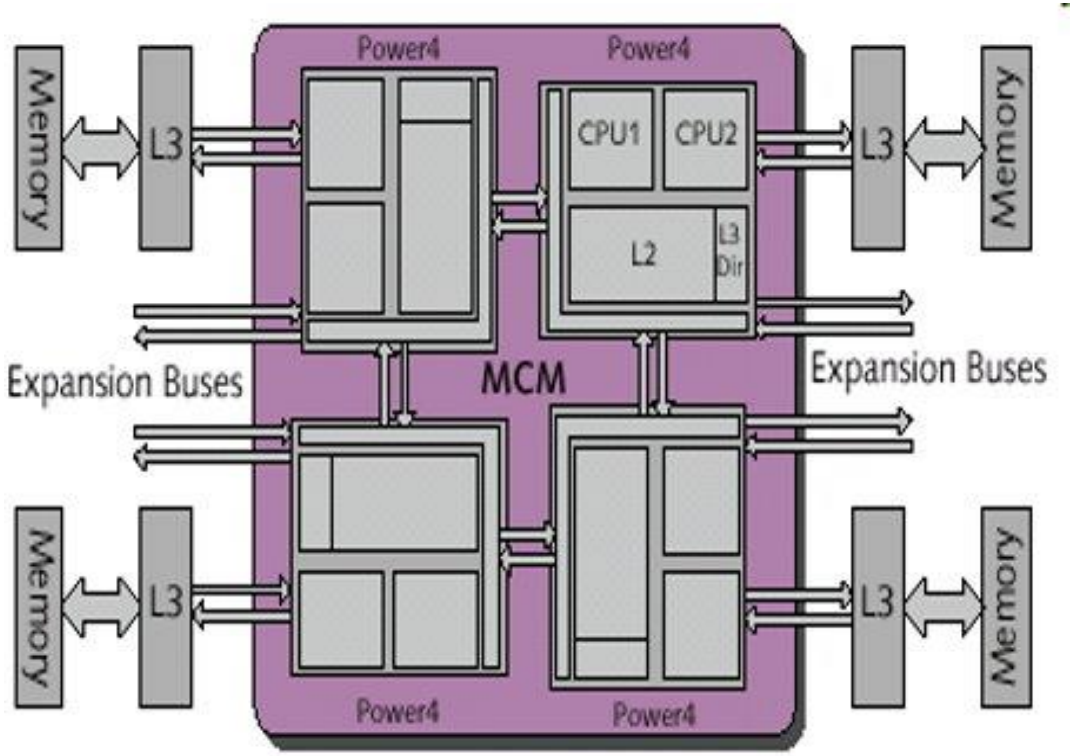
Multi-core

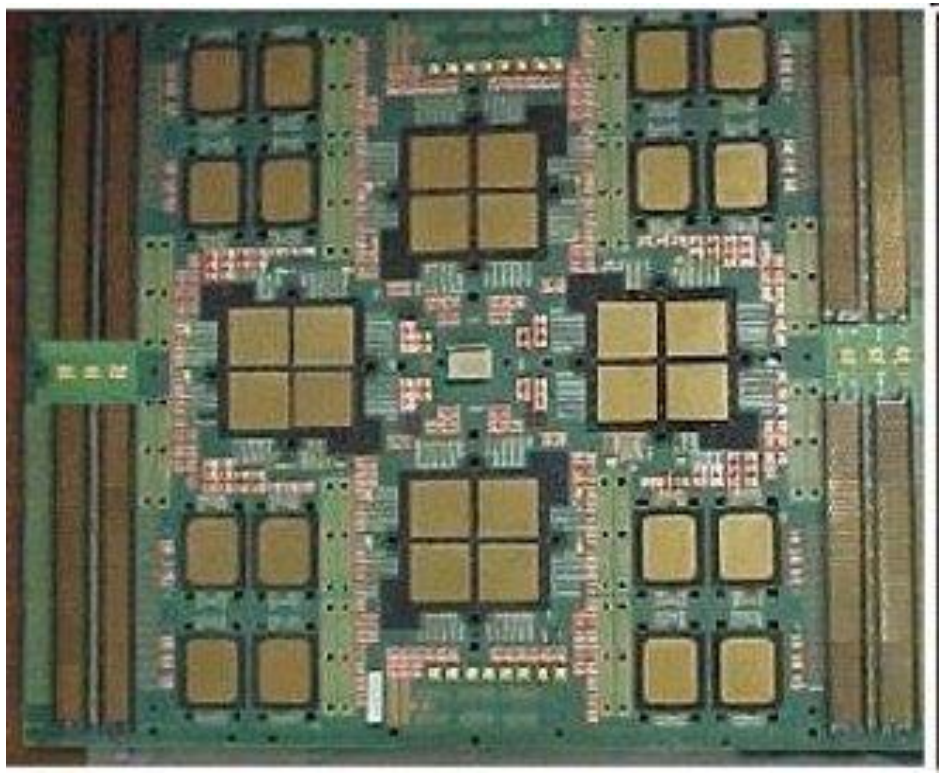


- Core = unit that reads and executes program instructions
- Processors were originally developed with only one core
- A multi-core processor is a single computing component with two or more independent processors ("cores").
- Replicate multiple processor cores on a single die.
- Manufacturers typically integrate cores onto a single integrated circuit die (chip multiprocessor or CMP), or onto multiple dies in a single chip package.

Parallel architectures

MultiChip Module





Accelerators

❑ Field-Programmable Gate Arrays

- ➢ A computer chip that can rewire itself for a given task
- ➢ Can be programmed with hardware description languages such as VHDL or Verilog

❑ General Purpose GPU

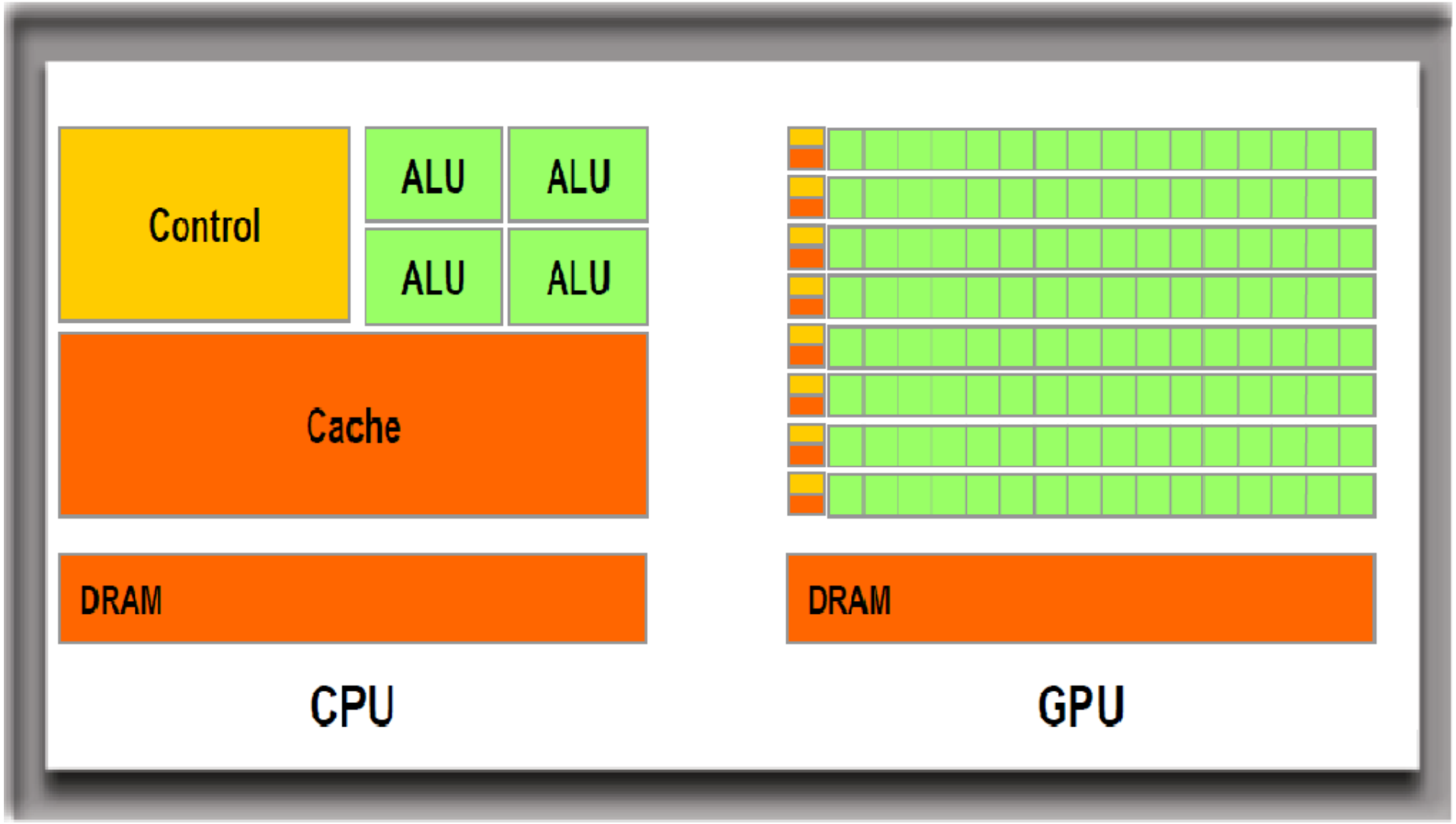
- ➢ General-purpose computing on graphics processing units (GPGPU)
- ➢ NVIDIA, Intel and AMD
- ➢ CUDA/OpenCL programming environment

NVIDIA V100


PCI Express 3.0 Host Interface
GigaThread Engine
GPC
TPC
SM
L2 Cache
Memory Controller
HBM2
High-Speed Hub
NVLink


SM
L1 Instruction Cache
L0 Instruction Cache
Warp Scheduler (32 thread/clk)
Dispatch Unit (32 thread/clk)
Register File (16,384 x 32-bit)
FP64
INT
FP32
TENSOR CORE
LD/ST
SFU
128KB L1 Data Cache / Shared Memory
Tex

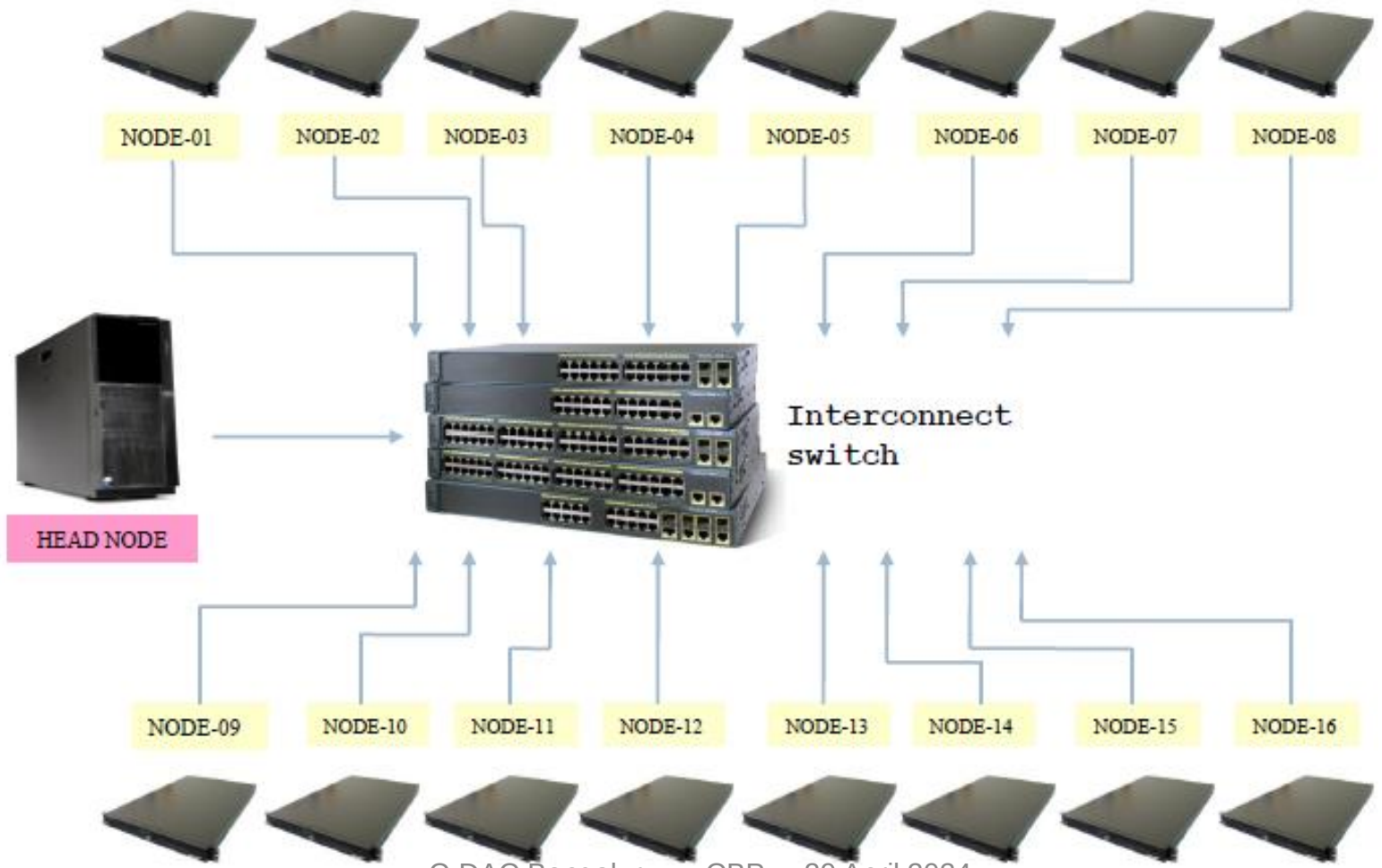
CPU vs GPU

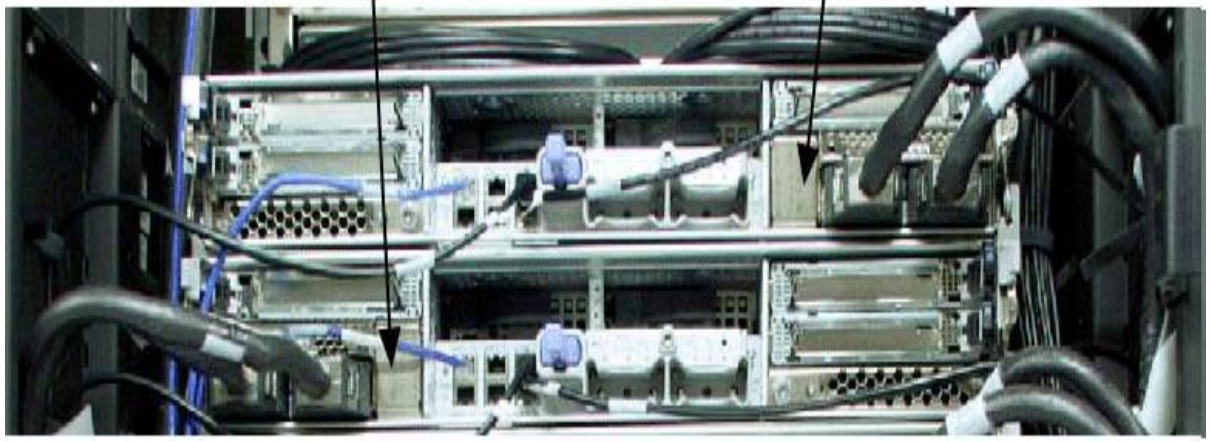


Cluster Computing

Clusters

Tightly coupled computers that work together closely as though they are a single computer






PARAM
padma
PARAM
padma
PARAM
padma

Cluster Computing Factors

Processing

- In general, there are two parallel processing approaches: symmetric multiprocessing (SMP) and massively parallel processing (MPP).

Interconnects

- In order for a large number of processors to work together, supercomputers utilize specialized network interfaces. These interconnects support high bandwidth and very low latency communication.
- Interconnects join nodes inside the supercomputer together.
- Popular Interconnects include GigE & Infiniband

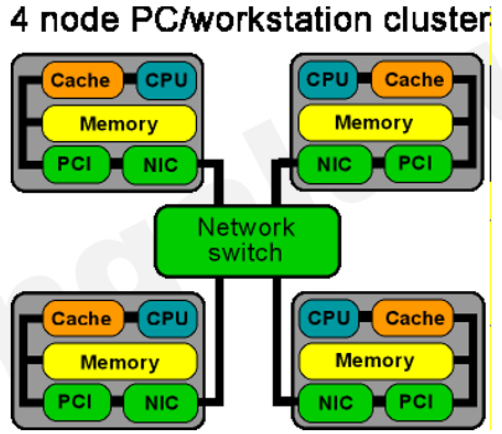
Operating Systems

- Supercomputers today most often use variants of the Linux operating system.
- "lightweight" operating systems that consist of a small, simple kernel with many of the capabilities of a general-purpose O/S removed.

Programming

- The parallel architectures of supercomputers demand the use of special programming techniques to exploit their speed.
- The base language of supercomputer code is, in general, Fortran or C, using special libraries to share data between nodes.
- Environments such as MPI and PVM for loosely connected clusters and OpenMP for tightly coordinated shared memory machines are used.
- GPGPUs have hundreds of processor cores and are programmed using programming models such as CUDA and OpenCL.

Other Parallel Systems



❑ Distributed Computing

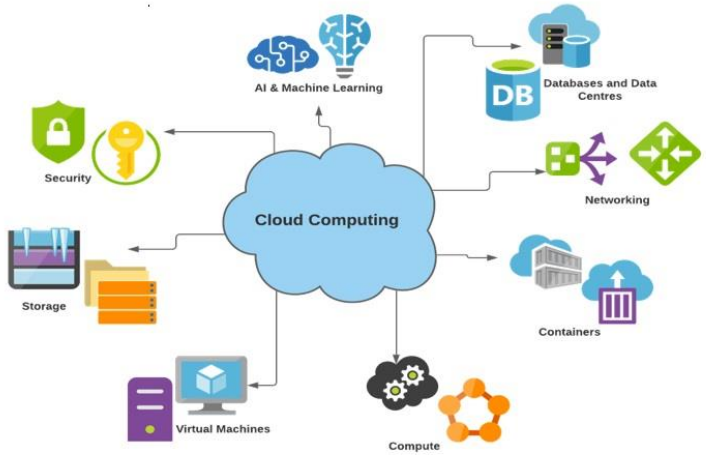
- ➢ Parts of a program are run at the same time, on separate computers communicating over a network

❑ Grid Computing

- ➢ Aggregation, sharing of distributed heterogeneous system

❑ Cloud Computing

- ➢ On-demand available service – IaaS, PaaS, SaaS
- ➢ Pay-as-you-use over the internet

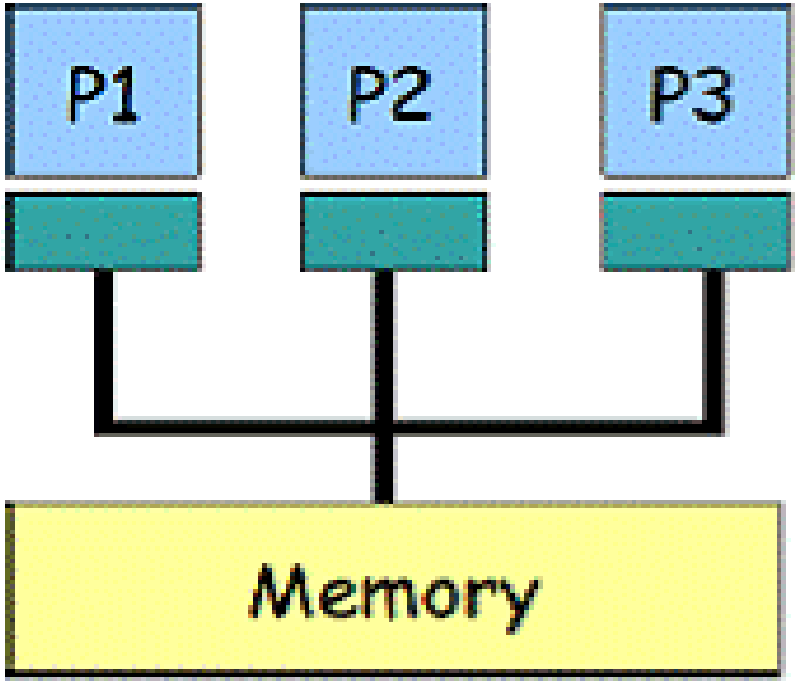


Memory Architecture

Shared Memory Architecture

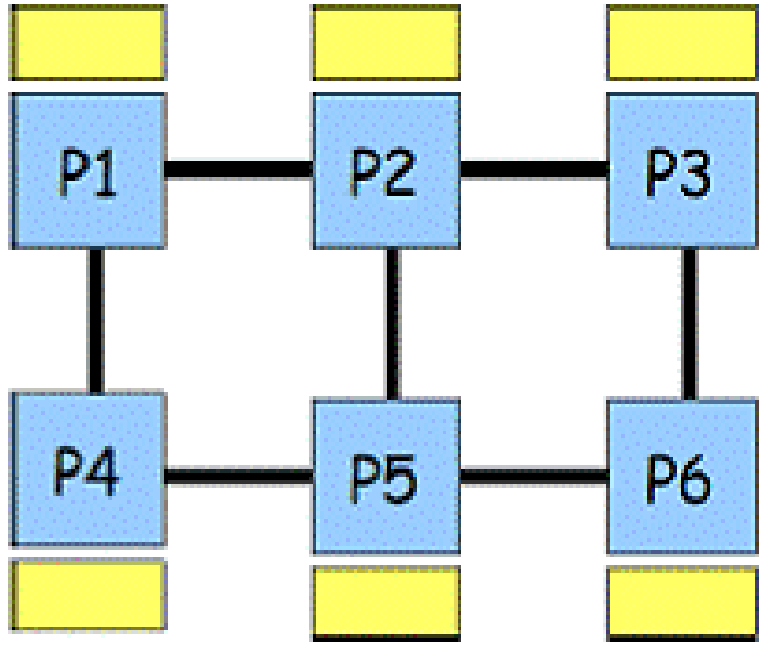
Uniform Memory Access (UMA):

- Example - SMP
- Identical processors
- Equal access and access times to memory
- Sometimes called Cache Coherent UMA (CC-UMA). Cache coherency is accomplished at the hardware level
- Contention - as more CPUs are added, competition for access to the bus leads to a decline in performance
  - ✓ Thus, scalability ~ 32 processors



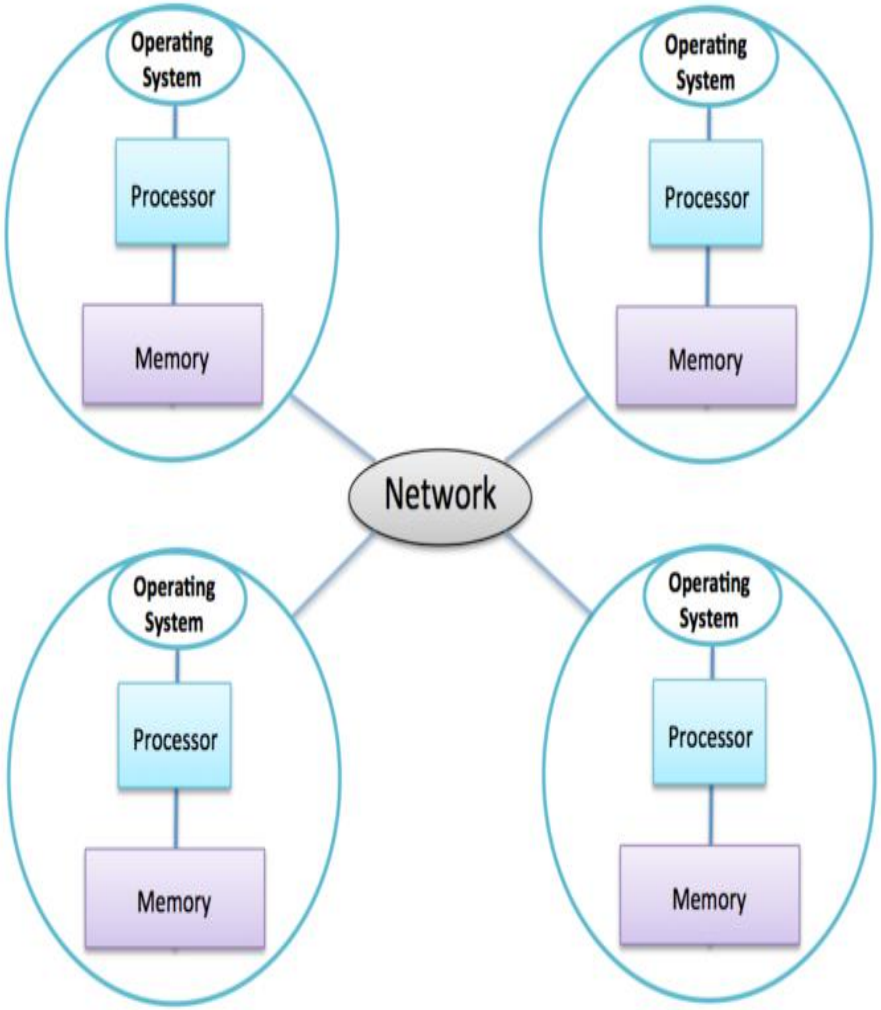
Shared Memory Architecture

- **Non-Uniform Memory Access (NUMA):**
  - Not all processors have equal access time to all memories
  - Memory access across link is slower
  - If cache coherency is maintained, then called CC-NUMA
  - Designed to overcome scalability, limitation of SMPs
  - Can support up to 1024 processors
  - Processors directly attached to a memory module experience lower latency than those attached to "remote" memory modules.

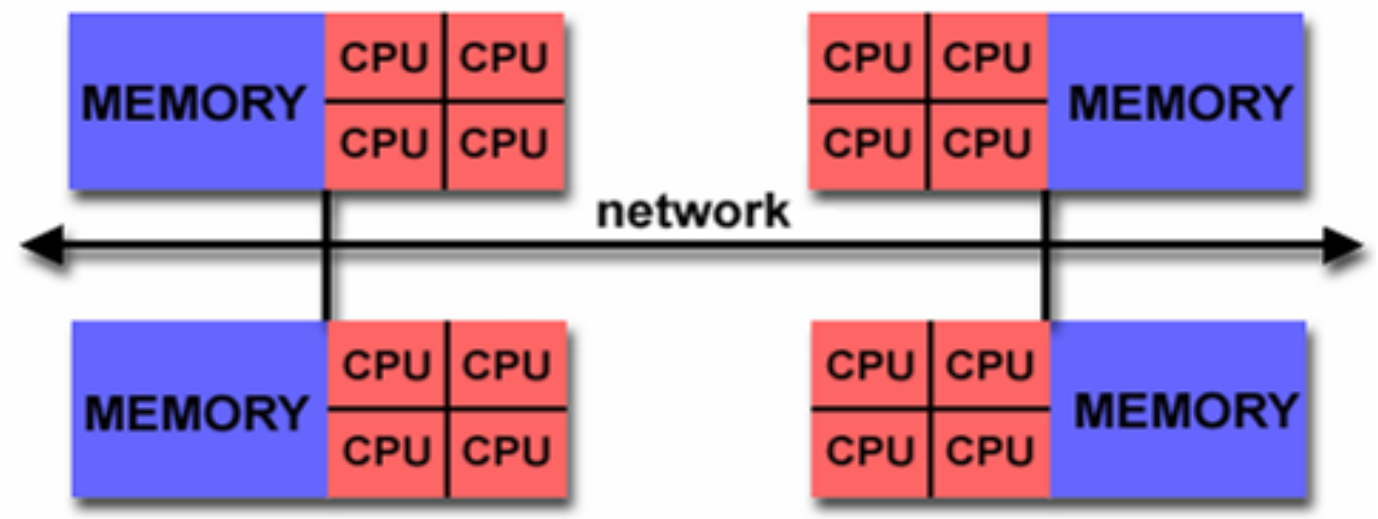


Distributed Memory Architecture

- Processors have their **own local memory**. So operates independently.
- **No** concept of **global address space** across all processors.
- Changes in local memory have no effect on memory of other processors. Hence, **cache coherency** does **not apply**
- **Data access** between processors is **defined by programmer** ; explicitly define how and when data is communicated. Synchronization between tasks is also programmer's responsibility.
- The network "fabric" used for data transfer varies widely, though it can be as simple as Ethernet.



Hybrid architecture

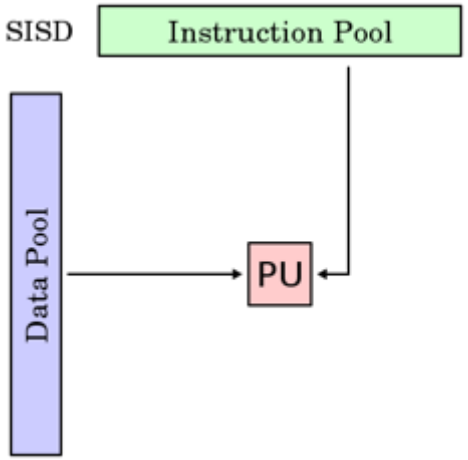


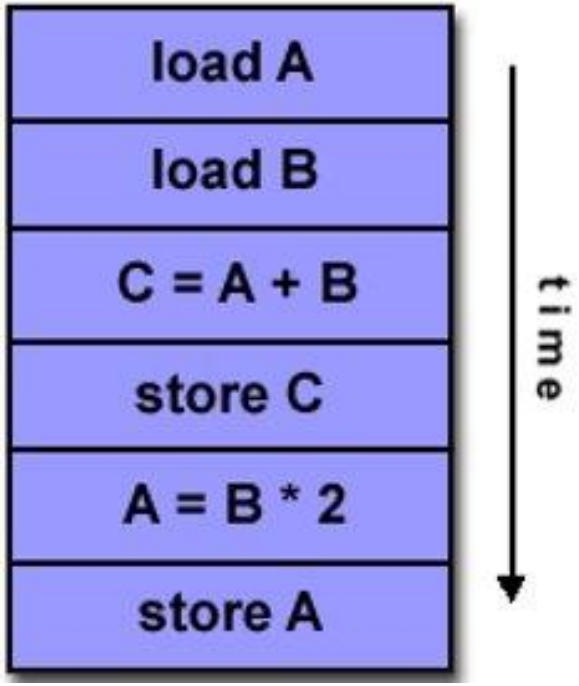
Parallel Computing Classification

Flynn's Taxonomy

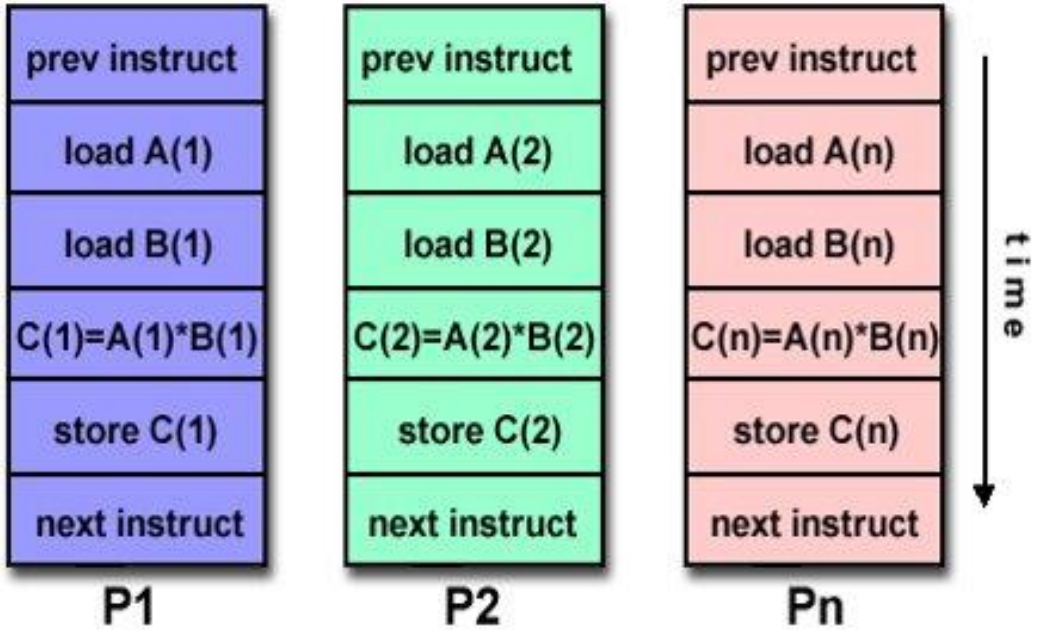
S I S D Single Instruction, Single Data	S I M D Single Instruction, Multiple Data
M I S D Multiple Instruction, Single Data	M I M D Multiple Instruction, Multiple Data

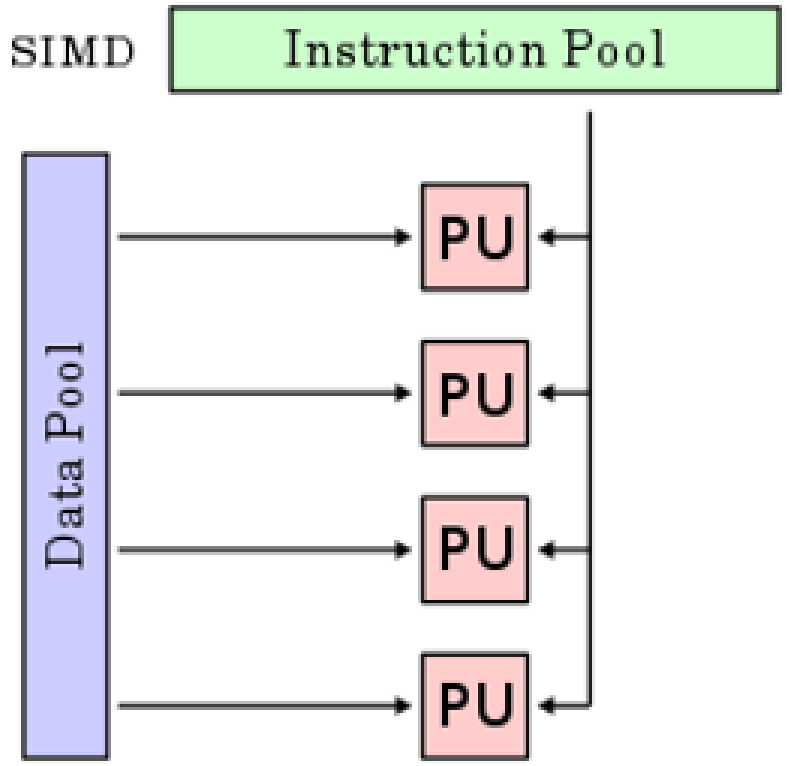
SISD


SISD
Instruction Pool
Data Pool
PU


load A
load B
C = A + B
store C
A = B * 2
store A
time

SIMD

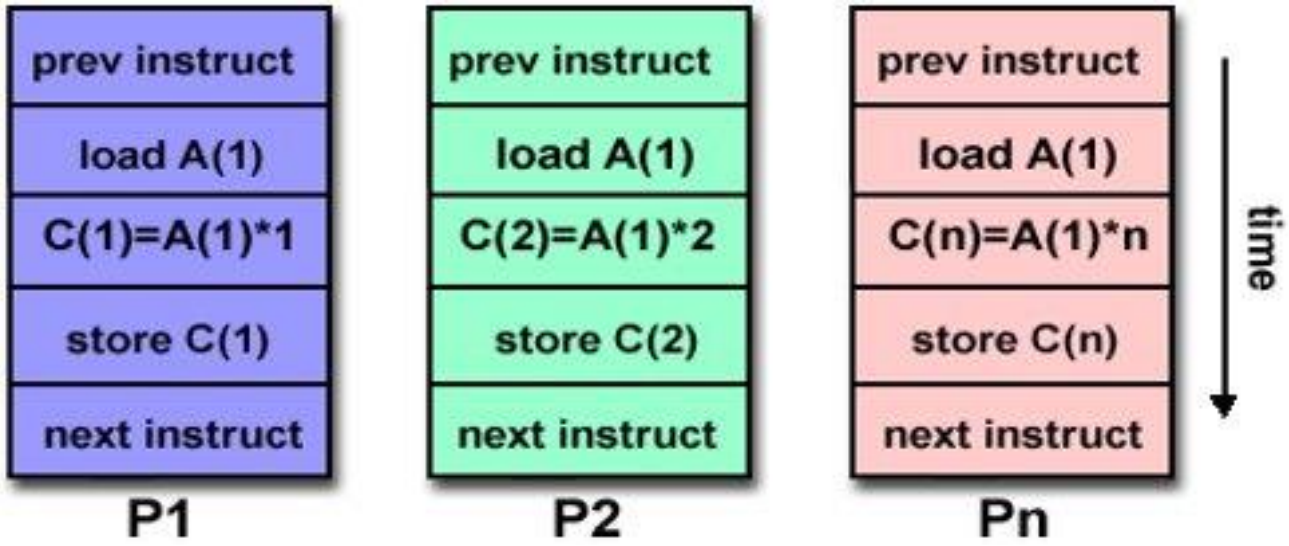


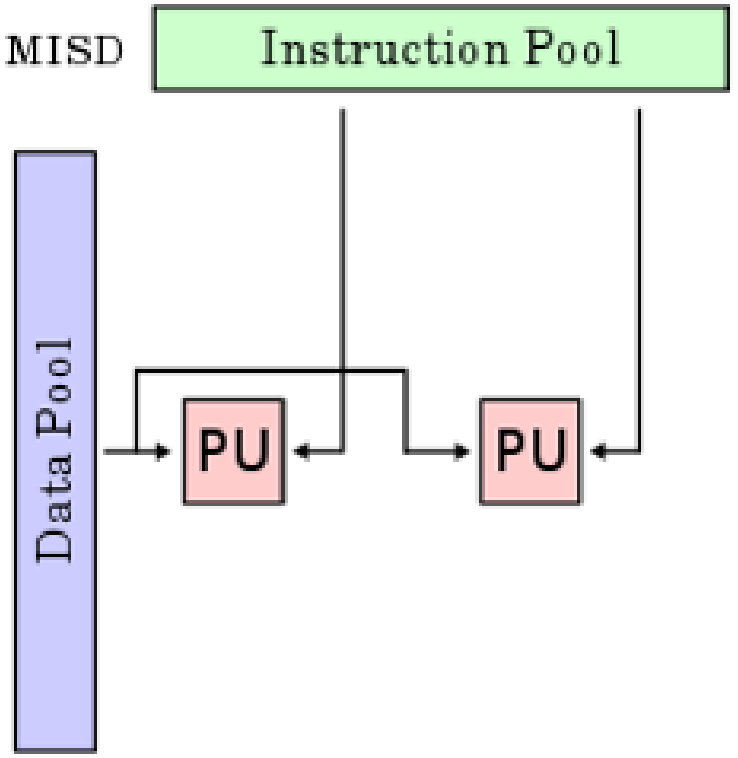


Processor Arrays: Connection Machine CM-2, MasPar MP-1 & MP-2, ILLIAC IV
Vector Pipelines: IBM 9000, Cray X-MP, Y-MP & C90, Fujitsu VP, NEC SX-2, Hitachi S820, ETA10

Most modern computers, particularly those with graphics processor units (GPUs) employ SIMD instructions and execution units.

MISD

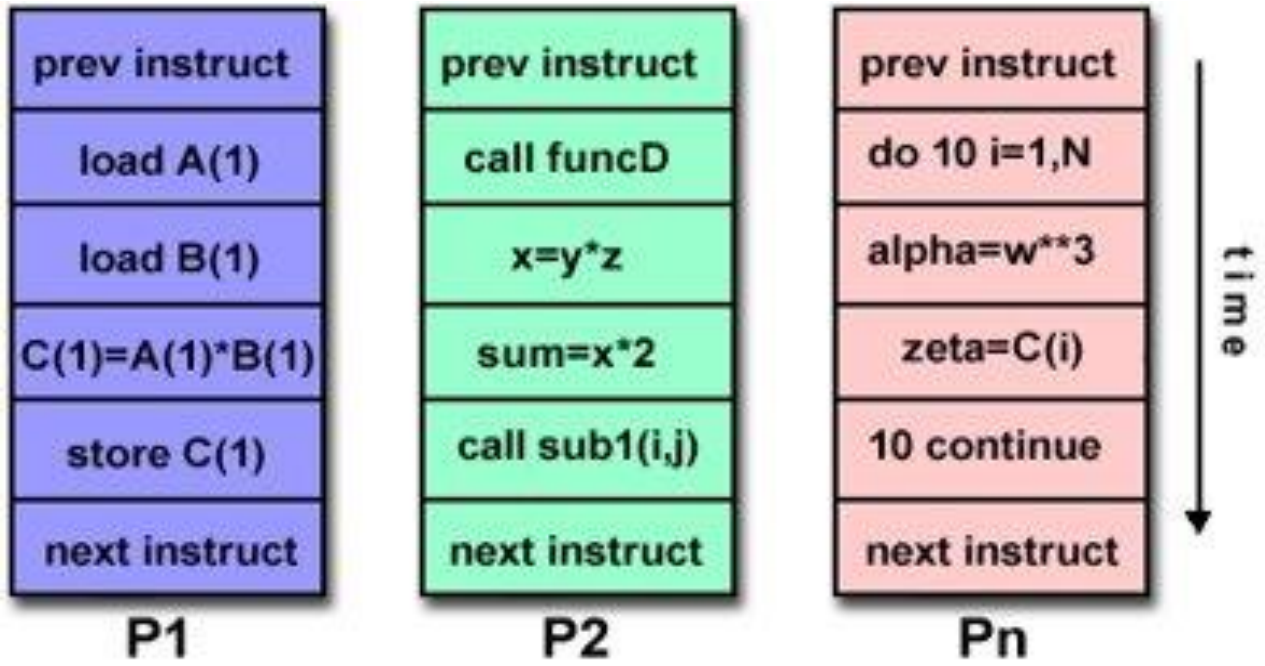


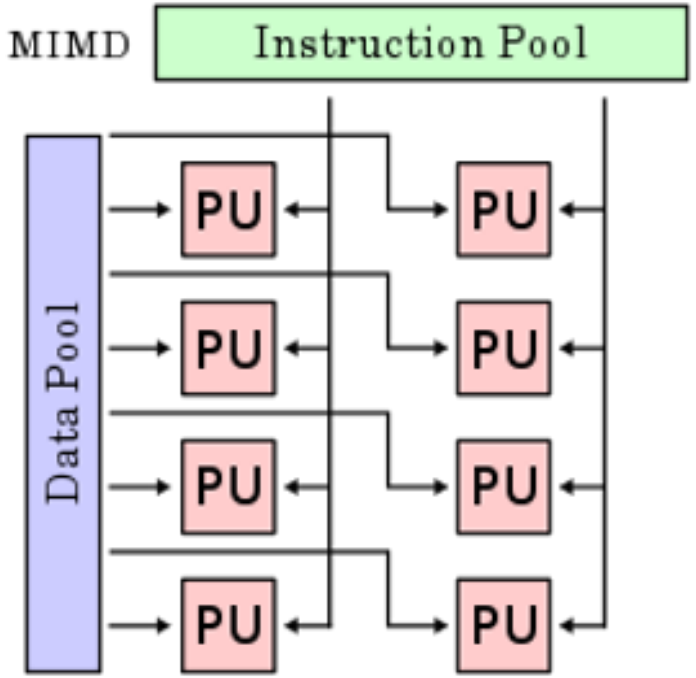


Some conceivable uses:

- multiple frequency filters operating on a single signal stream
- multiple cryptography algorithms attempting to crack a single coded message.

MIMD



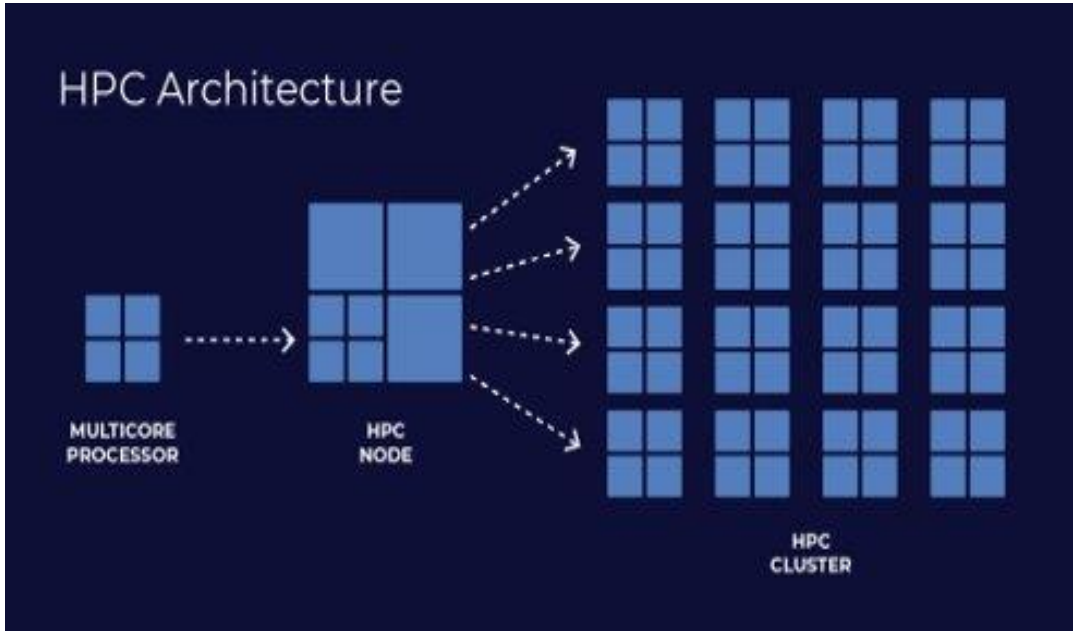


Most current supercomputers, networked parallel computer clusters and "grids", multi-processor SMP computers, multi-core PCs.

High Performance Computing

High Performance Computing

- HPC refers to the use of powerful computing systems to solve complex, data-intensive problems efficiently.
- It accelerates scientific research, engineering simulations, and data analysis, leading to breakthroughs in various fields.
- HPC leverages parallelism, harnessing multiple processors or nodes for rapid computations.
- Supercomputers are at the heart of HPC, delivering extraordinary computational power.



Units of HPC

- 1 Mflop/s 10^6
- 1 Gflop/s 10^9
- 1 Tflop/s 10^{12}
- 1 Pflop/s 10^{15}
- 1 Eflop/s 10^{18}
- 1 Zflop/s 10^{21}
- 1 Yflop/s 10^{24}

Weather Forecasting

If 1 cell is of size 1km x 1km x 1km
The whole atmosphere about 5 x 10^8 cells

If each calculations require 200 Flops,

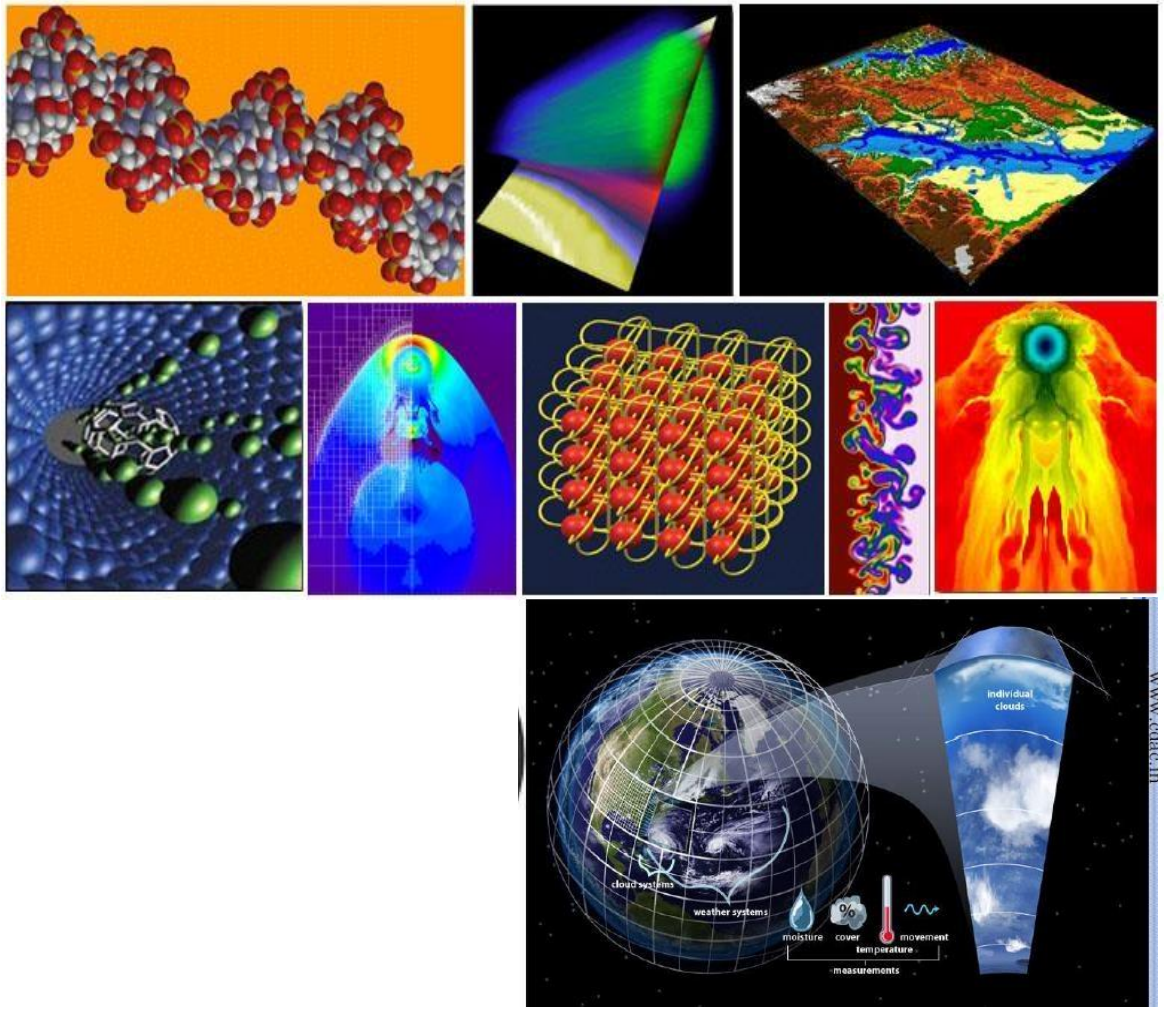
- it will take 200 FLOPS/cell * 5x10^8 cells for each time step
- Number of time steps for a 10-day simulation with 10-minute intervals (144 intervals/day) will be 1440

- Total FLOPS = (Number of FLOPS per time step per cell) * (Number of cells) * (Number of time steps)
- Total FLOPS = (200 FLOPS/cell) * (5x10^8 cells) * (1440 time steps)
- Total FLOPS = 1.44 x 10^17 FLOPS

- **It would take approximately 1,666 days to compute the simulation with a 100 MFLOPS system and about 20 hours to compute the simulation with a 2 TFLOPS system.**

HPC Applications

- Prediction of weather, climate, global changes
- Challenges in materials sciences
- Semiconductor design
- Structural biology
- Design of drugs
- Human genome
- Astronomy
- Challenges in transportation
- Vehicle dynamics
- Nuclear fusion
- Enhanced oil and gas recovery
- Computational ocean sciences
- Speech
- Vision
- Visualization and graphics



Supercomputer

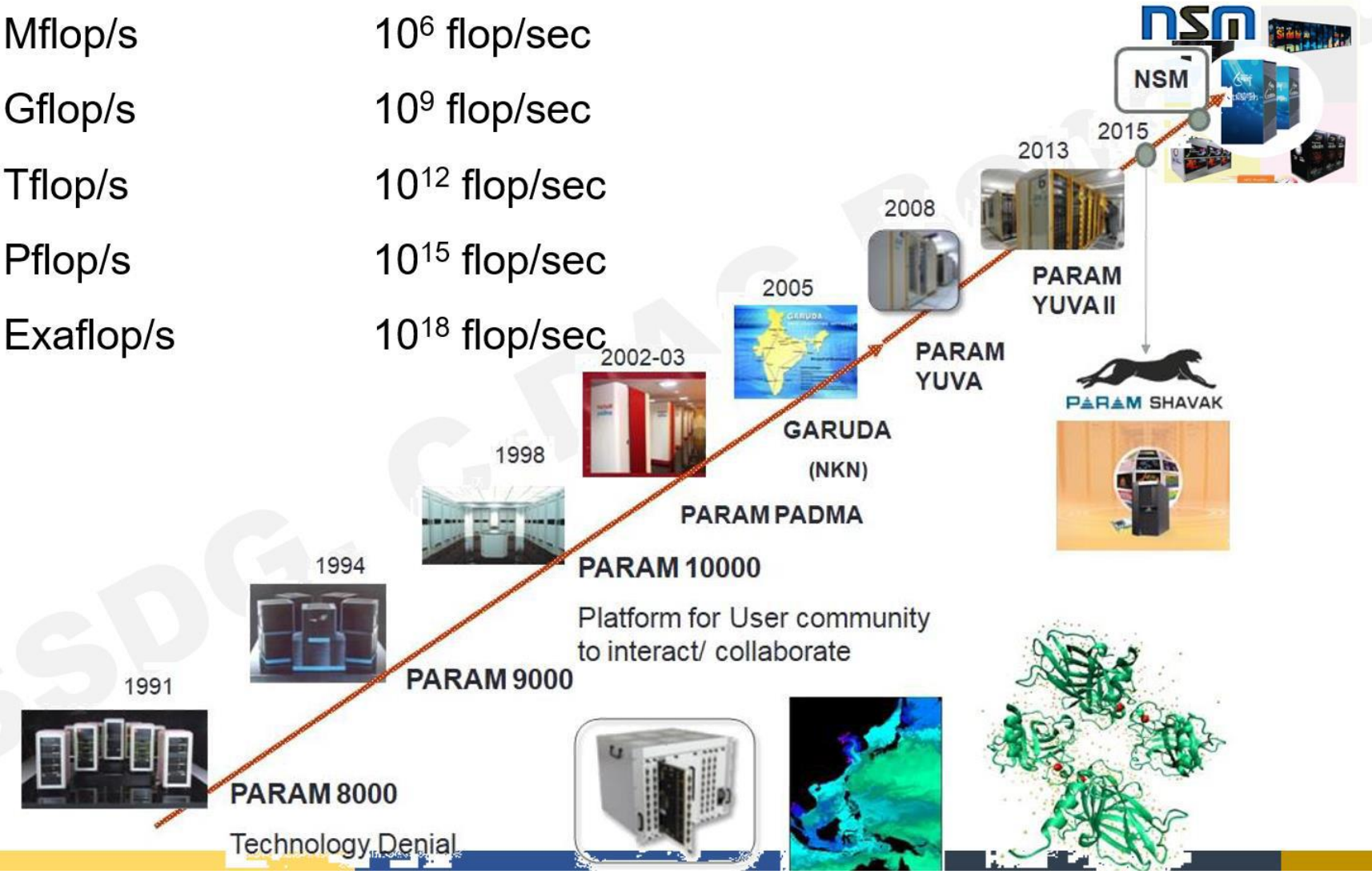
- Supercomputers are a class of high-performance computing (HPC) systems designed to offer higher computational power and processing speed.
- Tightly coupled computers that work together closely as though they are a single c omputer

World Top5 Supercomputers

Rank	Site	System	Cores	Rmax (PFlop)	RPeak (PFlop)
1	Frontier , DOE/SC/Oak Ridge National Laboratory United States	HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE	8,699,904	1,194.00	1,679.82
2	Aurora, DOE/SC/Argonne National Laboratory United States	HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel	4,742,808	585.34	1,059.33
3	Eagle, Microsoft Azure United States	Microsoft NDv5, Xeon Platinum 8480C 48C 2GHz, NVIDIA H100, NVIDIA Infiniband NDR	1,123,200	561.20	846.84
4.	A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu	A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu	7,630,848	442.01	537.21
5.	LUMI, EuroHPC/CSC Finland	HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE	2,220,288	309.10	428.70

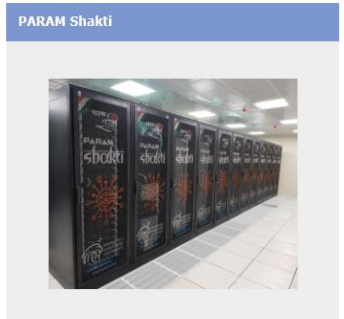
Indian/C-DAC line of Supercomputers

Mflop/s	10^6 flop/sec
Gflop/s	10^9 flop/sec
Tflop/s	10^{12} flop/sec
Pflop/s	10^{15} flop/sec
Exaflop/s	10^{18} flop/sec



NSM Sites

- PARAM Shivay, IIT BHU, 837 TFLOPS
- PARAM Shakti, IIT Kharagpur, 1.66 PFLOPS
- PARAM Brahma, IISER Pune, 797 TFLOPS
- PARAM Yukti, JNCASR Bengaluru, 838 TFLOPS
- PARAM Sanganak, IIT Kanpur, 1.67 PFLOPS
- PARAM Pravega, IISc Bangalore, 3.3 PFLOPS
- PARAM Seva, IIT Hyderabad, 838 TFLOPS
- PARAM Smriti, NABI, Mohali, 838 TFLOPS
- PARAM Utkarsh, C-DAC Bengaluru, 838 TFLOPS
- PARAM Siddhi-AI, C-DAC Pune,
- PARAM Ganga, IIT Roorkee, 1.67 PFLOPS
- PARAM Ananta, IIT Gandhinagar, 838 TFLOPS
- PARAM Porul, NIT Trichy, 838 TFLOPS
- PARAM Himalaya, IIT Mandi, 838 TFLOPS
- PARAM Kamrupa, IIT Guwahati, 838 TFLOPS

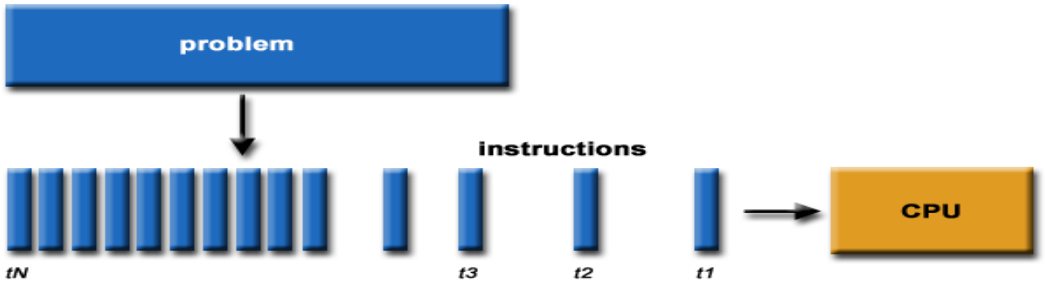


Indian Top5 Supercomputers
(https://topsc.cdacb.in)

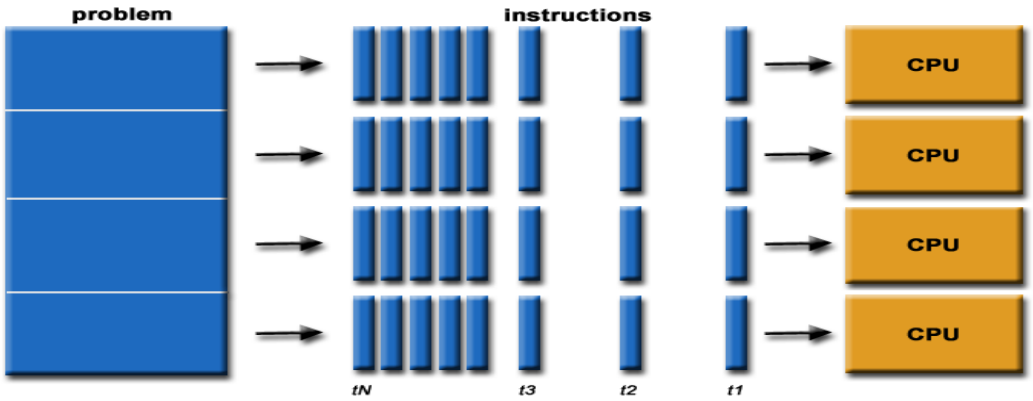
Rank	Site	System	Cores/ nodes	Rmax (TFlop)	RPeak (TFlop)
1 [90]	Airawat-PSAI, C-DAC Pune	NVIDIA DGX A100, AMD EPYC 7742 64C 2.25GHZ, NVIDIA A100, INFINIBAND HDR2	81344	8500	13176
2 [163]	PARAM Siddhi-AI, C-DAC,Pune	NVIDIA DGX A100, AMD EPYC 7742 64C 2.25GHz, NVIDIA A100, Mellanox HDR Infiniband (OEM:ATOS under NSM initiative, Bidder)	41664/ 236	4619	5267.14
3 [201]	Pratyush, Indian Institute of Tropical Meteorology, Pune	Cray XC-40 class system with 3315 CPU-only (Intel Xeon Broadwell E5-2695 v4 CPU) nodes with Cray Linux environment as OS, and connected by Cray Aries interconnect	119232/ --	3763.9	4006.19
4. [354]	Mihir, NCMRWF, Noida	Intel Xeon Broadwell E5-2695 v4 CPU with Cray Linux environment connected by Cray Aries interconnect OEM:Cray, Bidder:Cray	83592/ 1152	2570.4	2808.7
5.	PARAM Pravega, IISc, Bangalore	Intel Xeon Cascade Lake processors,NVIDIA Tesla V100 with NVLink, Mellanox HDR interconnect. OEM:Atos, Bidder:Atos	29952/ 624	1702	2565

Parallel Programming

What is Parallel Programming



Serial



Parallel

- A parallel processing program is a single program that runs on multiple processors simultaneously.
- The overall problem is split into parts, each of which is performed by a separate processor in parallel.
- In addition to faster solution, it may also generate a more precise solution.

Parallel Programming

- ✓ Identifying inherent parallelism
- ✓ Partitioning/Decomposition
- ✓ Dependency analysis
- ✓ Programming Paradigms
- ✓ Performance

Identification of Parallel region

- Identifying the parallelizable parts in a program is a crucial step in parallel programming.
- It involves analyzing the program's structure, algorithms, and data dependencies to identify portions that can be executed concurrently.
- Some techniques and approaches to identify parallelism are :
  - Manual Inspection
  - Static Code Analysis
  - Code Profiling

Identification (Contd.)

- Example:

```
int i, a[], b[], c[], x, t, y;
x=t-y;
for(i=0;i<100;i++)
    a[i]=b[i]*c[i];
y=t+x;
```

Parallelizable

Decomposition

- Decomposition in parallel programming refers to the process of breaking down a problem or task into smaller, manageable parts that can be executed concurrently.
- Mainly there are 4 decomposition methods:
  - Data decomposition
  - Recursive decomposition
  - Exploratory decomposition
  - Speculative decomposition

Data Decomposition

- Here the decomposition of computations is done in two steps:
  - Data partition and Task creation

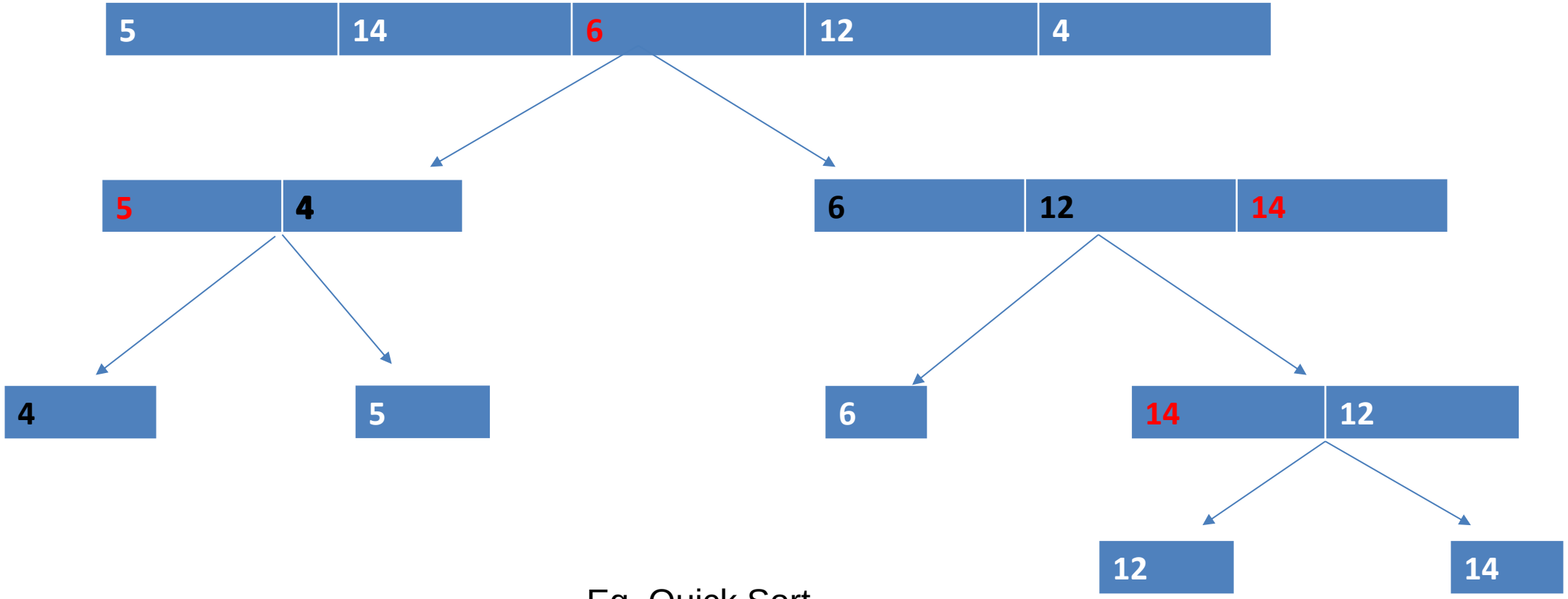
- Example:

$$\begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} + \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix} = \begin{bmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{bmatrix}$$

Task 1: $C_{1.1} = A_{1.1}+B_{1.1}$
Task 2: $C_{1.2} = A_{1.2}*B_{1.2}$
Task 3: $C_{2.1} = A_{2.1}*B_{2.1}$
Task 4: $C_{2.2} = A_{2.2}*B_{2.2}$

Recursive decomposition

A problem is divided into smaller subproblems of the same type, and the same approach is recursively applied to each subproblem until a base case is reached. It is based on the principle of divide and conquer



Eg. Quick Sort

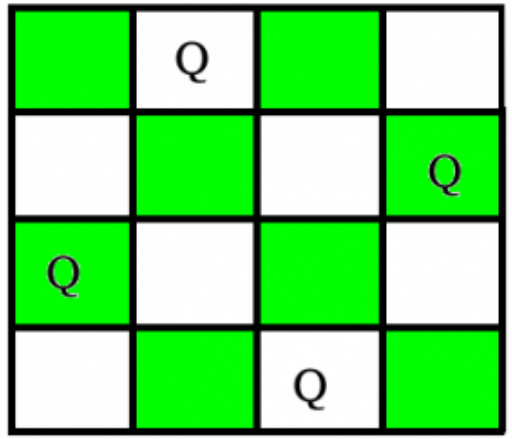
Exploratory Decomposition

- The idea behind exploratory decomposition is to dynamically explore and adapt the decomposition strategy based on insights gained during the execution of the parallel program. It allows for flexibility and adaptability in finding the most efficient decomposition of the problem.

- These problems typically involve the exploration (search) of a state space of solutions.

- Problems in this class include a variety of discrete optimization problems, theorem proving, game playing, etc.

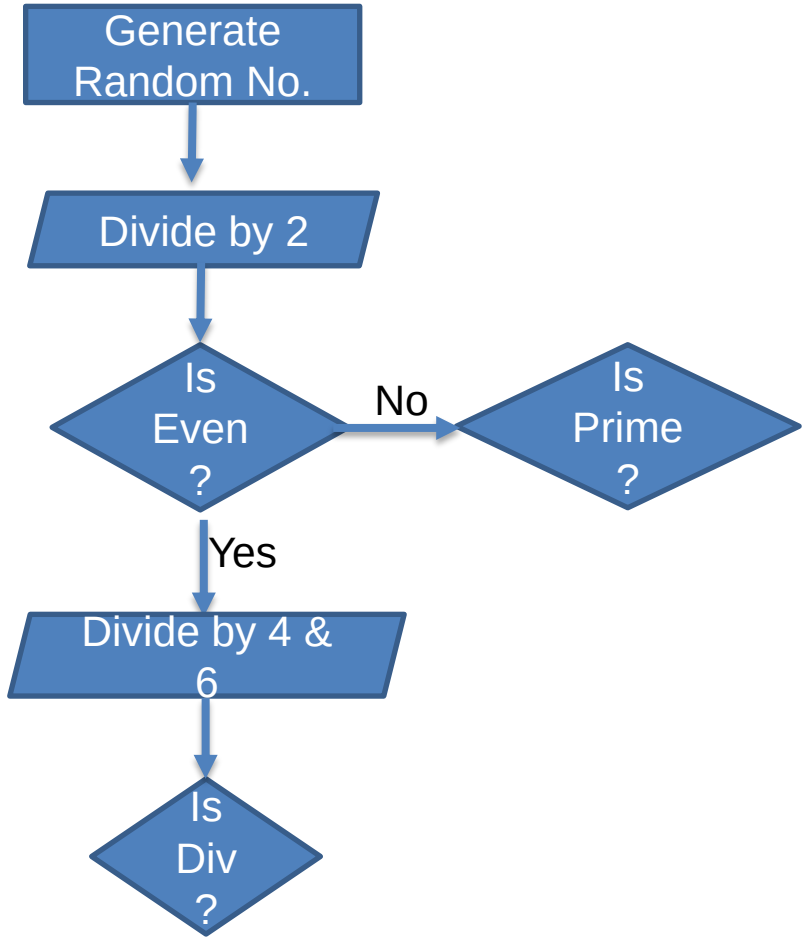
- Example: n Queens problem



Speculative Decomposition

- This decomposition is used to exploit potential parallelism by speculatively executing multiple independent tasks or subproblems concurrently, without knowing for certain if all of them are necessary or will lead to a valid solution.

- While one task is performing the computation whose output is used in deciding the next configuration, other tasks can concurrently start the computations of the next stage.

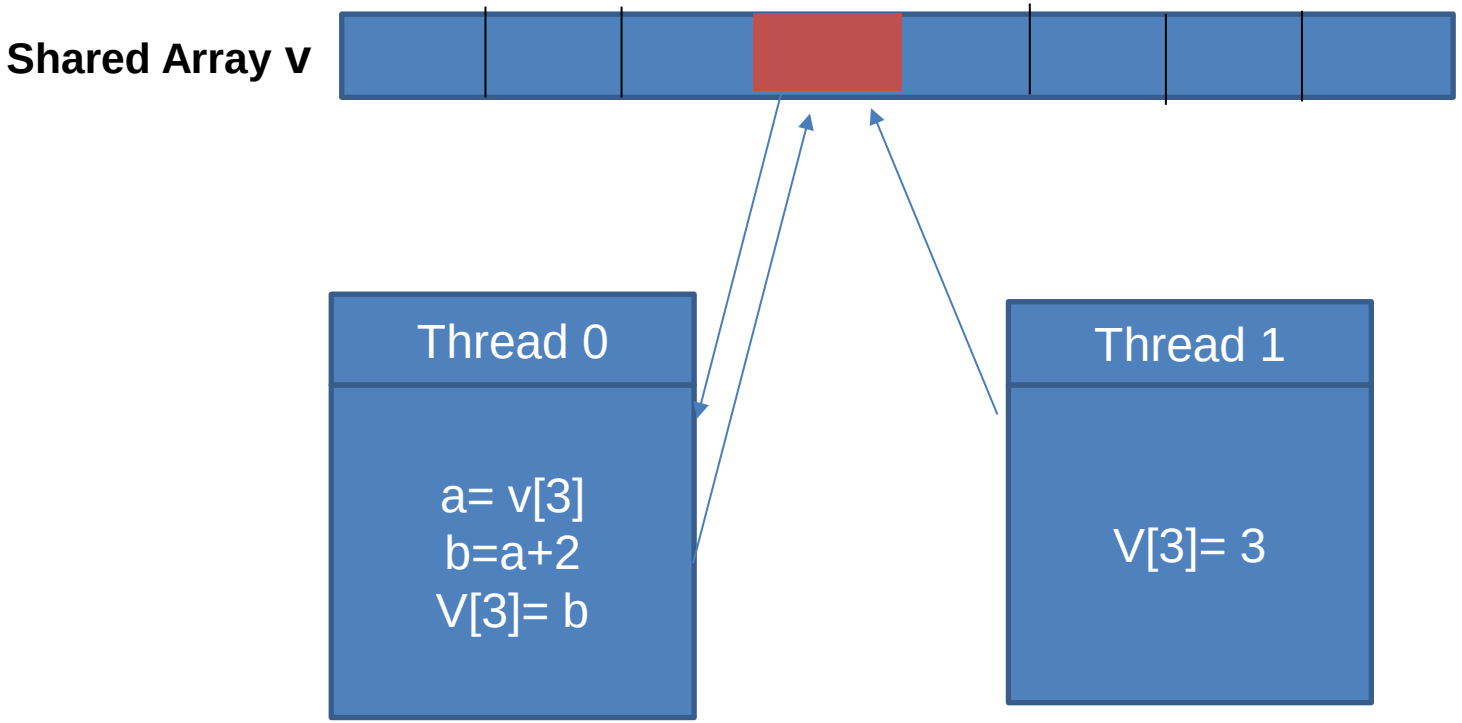


Data Dependency

- In parallel programs, order of statements in the code may not match the execution order;
  - data dependencies must be recognized and ensure that the computations are executed in the correct order.
- Data dependencies can be managed by
  - isolating dependencies within individual tasks or
  - coordinating the execution of multiple tasks.
- When data dependencies exist across parallel tasks, we do explicit coordination in execution of the task.

Data Dependency

This is an example of a *race condition* because two or more threads access the same location in memory and at least one of the threads writes to the shared location.



Even if the programmer intends these read and write operations to occur in a particular order, no steps were taken to manage the data dependency.

Data Dependency

Combined Thread ops	Combined Thread ops	Combined Thread ops
T1: V[3] = 3 T0: a = v[3] T0: b = a+2 T0: v[3]= b	T0: a = v[3] T1: V[3] = 3 T0: b = a+2 T0: v[3]= b	T0: a = v[3] T0: b = a+2 T0: v[3]= b T1: : V[3] = 3
Result	Result	Result
V[3]= 5	V[3]= 2	V[3]=3

- ✓ We might receive a different outcome every time the program runs if there is no mechanism.
- ✓ If the second thread (thread 1) writes a value into the vector and we want the first thread (thread 0) to use it, we need to use a technique to make sure thread 1 wrote the value before thread 0 read it.

Data Dependency

- Three types of data dependency or data hazards:
  - Read-after-Write (RAW)
  - Write-after-Read (WAR)
  - Write-after-Write (WAW)

RAW dependency

- Occurs when value produced by instruction is required by an subsequent instruction.
- Also known as true dependency or flow dependency
- Example:
  - ADD **R1**, R2, R3
  - SUB R4, **R1**, 5

- They are one of the most difficult dependencies to resolve in hardware. (unparallelizable)

WAR dependency

- Occurs when an instruction writes to a location which has been read by a previous instruction
- Example:
  - ADD R3, **R2**, R1
  - SUB **R2**, R5, 1
- Instruction 2 must not produce its result in R2 before instruction 1 reads R2 , Otherwise instruction 1 would use the value produced by instruction 2 rather than the previous value of R2.

  Also called name dependency, as they can be removed through renaming of register/variables

```
ADD R3, R2, R1
SUB R7, R5,1
```

Output Dependency

- Occurs when a location is written by two instruction
- Example:
  - ADD **R3**, R2, R1
  - SUB R2, **R3**, 1
  - ADD R3, R2, R5
- Instruction 1 must produce its result in R3 before instruction 3 produces its result in R3 otherwise instruction 2 might use the wrong value of R3.
- Output Dependency are form of resource conflict (register reuse) and can be resolved by register renaming.

Loop carried Dependence

When a memory location is read or written in one iteration and written in another iteration.

- Loop carried dependencies – occurs between accesses across different loop iterations.
- Loop independent dependence - occurs between accesses in the same loop iteration

Example:

```
for (i=0; i<n; i++)
 {
   A[i]=B[i];
   A[i+1]=A[i]*2;
}
```

Control flow dependency

- Execution of instructions depends on the outcome of a branch or control flow statement.
- Control dependencies restrict parallel execution because the outcome of one branch determines which instructions will be executed next

Example 1:

```
int x = 10; int y = 5; result;
If (x > y) {
        result = x - y; }
else {
        result = y - x; }
```

Example 2:

```
ADD  R1,  R2,  R3 ;
SUB  R4,  R5,  R6 ;
CMP  R4,  R1 ;
BEQ  Label ;
ADD R7,  R8,  R9 ;

Label: SUB R10, R11, R12 ;
```

Communication

When do processors communicate ?

- To share data between different parallel tasks or processes
- To manage control flow dependencies,

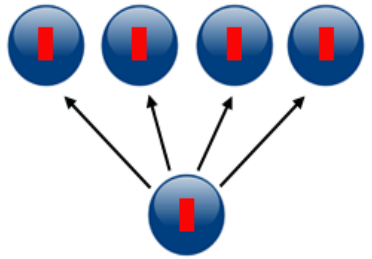
  E.g. when branching or making decisions based on results computed by other processors
- Communicating to share the error information to other processors for error handling or recovery
- Coordinate task dependencies, or assign subtasks to achieve efficient task coherence (Load Balance)

Communication

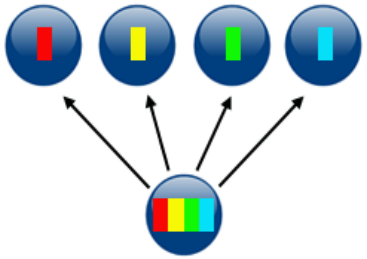
How do processors communicate ?

- In shared memory programming, use a semaphore or other locking mechanism for synchronization

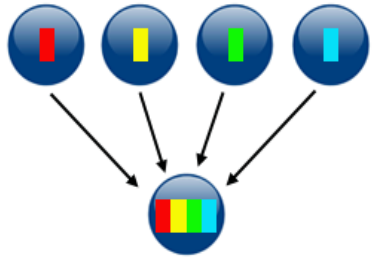
- In distributed memory programming, use messages to communicate the results of earlier calculations.



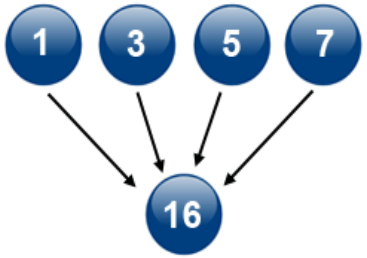
broadcast



scatter



gather



reduction

Tips: Dependent calculations should be grouped into the same thread of execution if possible.

Synchronization

- int main(…){

```
int num_t, tid;        //Shared
#pragma omp parallel
  num_t = omp_get_num_threads()
  tid = omp_get_thread_num();
  printf("Hi from thread %d of %d \n", tid, numt);
}
```

- What will this print?

Synchronization (contd.)

Normal scenario

Output 1:

Hi from thread 0 of 4
Hi from thread 1 of 4
Hi from thread 2 of 4
Hi from thread 3 of 4

Output 2:

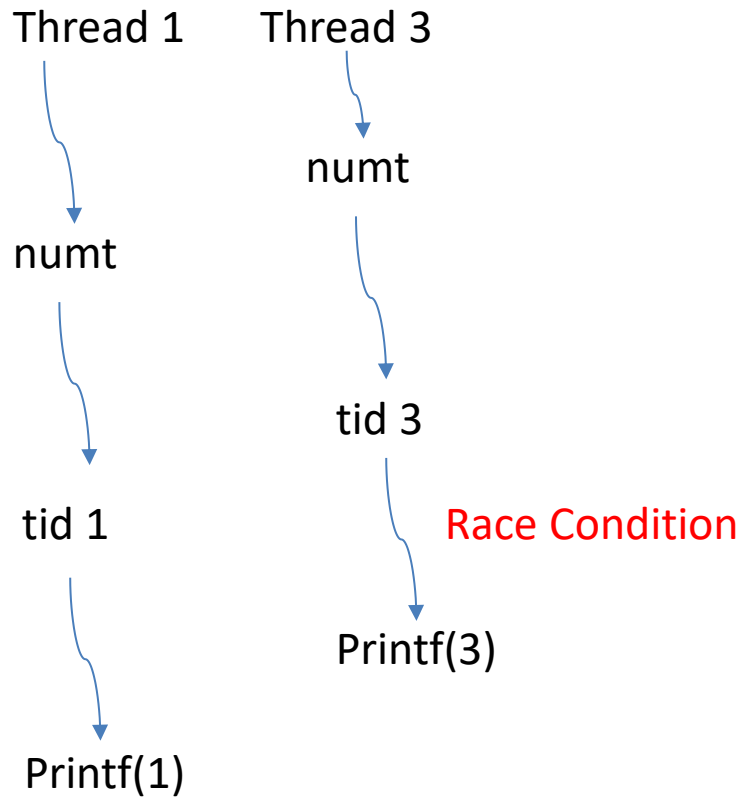
Hi from thread 2 of 4
Hi from thread 1 of 4
Hi from thread 0 of 4
Hi from thread 3 of 4

Synchronization (contd.)

If there is a race condition

Output:

Hi from thread 2 of 4
Hi from thread 1 of 4
Hi from thread 0 of 4
Hi from thread 1 of 4



Synchronization (Contd.)

We need a mechanism to intelligently split the execution of a program

```
int main(…){
        int num_t, tid;        //Shared
        #pragma omp parallel private(tid) shared(numt)
                num_t = omp_get_num_threads()
                tid = omp_get_thread_num();
                printf("Hi from thread %d of %d \n", tid, numt);
        }
```

- Synchronization overhead
- In case barrier is used, the slowest task determines the speed of the whole calculation.

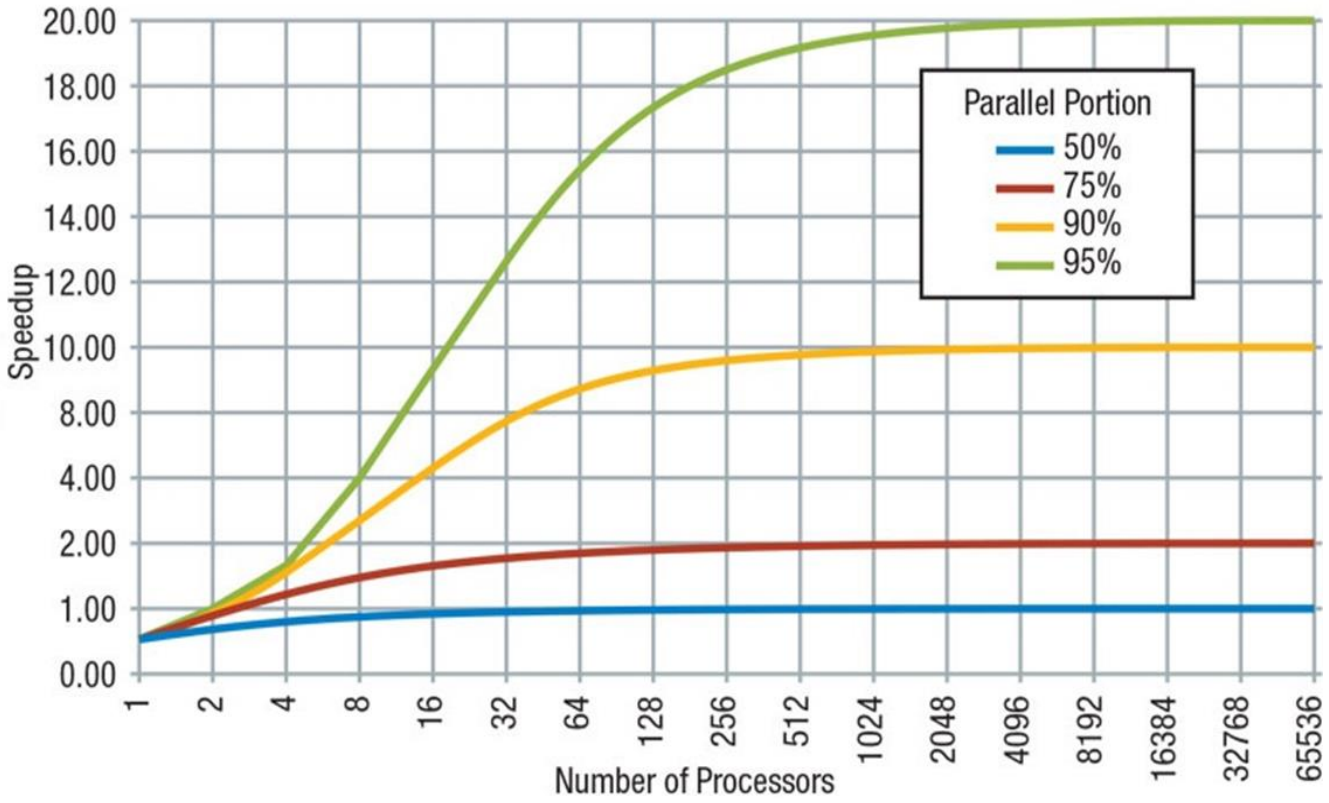
Granularity

- **Granularity** (or grain size) of a task is a measure of the amount of work (or computation) which is performed by that task.
- Or Granularity=computation time/communication time
- Two types of Granularity:
  - Fine-grained granularity :
    - Smaller ‘chunk’ size.
    - Relatively less computation work between communication events.
    - Better with shared memory model.
    - Less communication overhead.
    - Better choice while handling load balancing.

Granularity(Contd.)

- Coarse-grained granularity.
  - Bigger ‘chunk size’.
  - Computation to communication ratio is high.
  - Tasks completes more work with relatively less communication/synchronization events.
  - Well suited for distributed memory system.
  - Used in most cases for improved performance because of less communication requirement.

Amdahl’s law

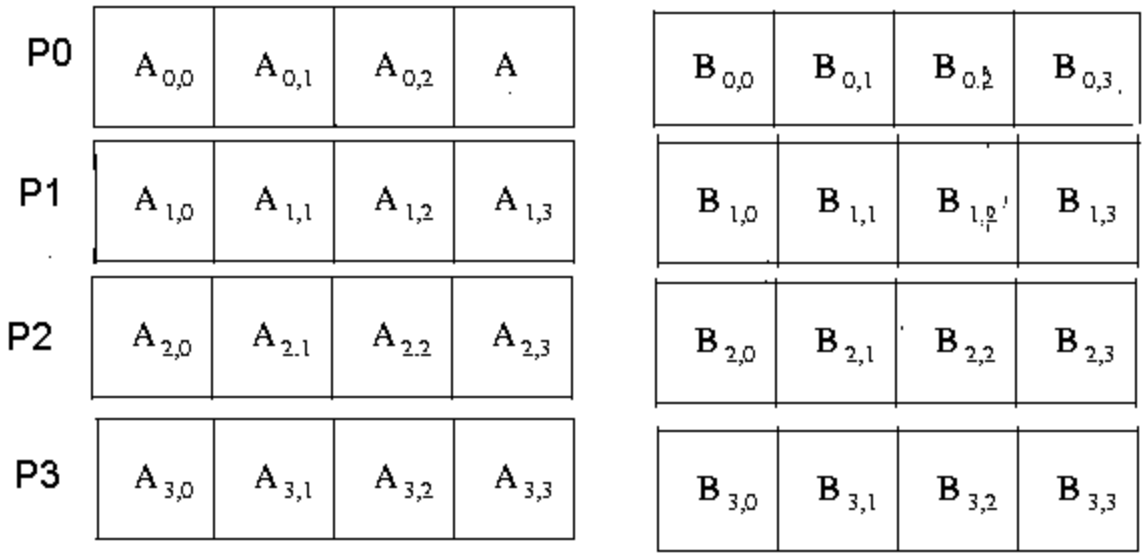


According to Amdahl's Law, the speedup of a program is limited by the fraction of the program that cannot be parallelized and applications can almost never be completely parallelized.

$$S(p) = \frac{1}{(1-P)+\frac{P}{N}},$$

Even if you have an infinite number of processors(n), there will always be a limit to the achievable speedup based on the serial portion of the program (1 - p).

Matrix-Matrix Addition



Approach For Parallelizing Matrix Matrix Addition

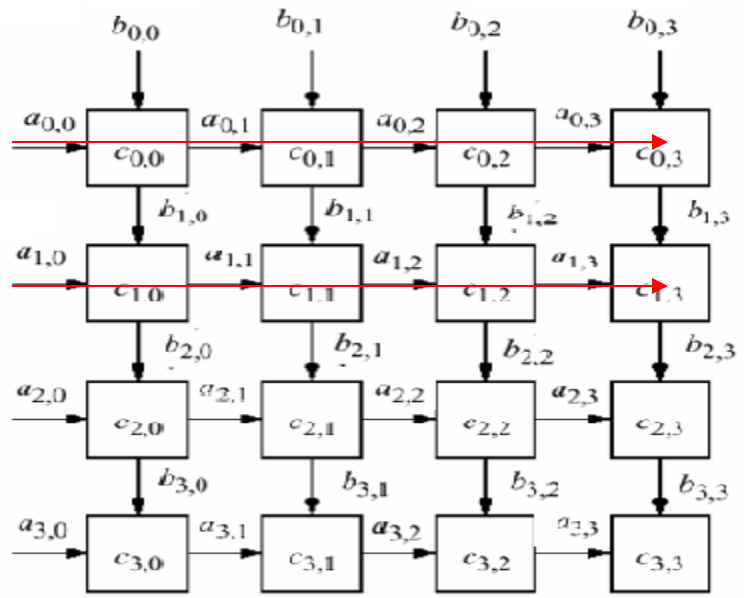
$A_{0,0}$	$A_{0,1}$	$A_{0,2}$	A
$A_{1,0}$	$A_{1,1}$	$A_{1,2}$	$A_{1,3}$
$A_{2,0}$	$A_{2,1}$	$A_{2,2}$	$A_{2,3}$
$A_{3,0}$	$A_{3,1}$	$A_{3,2}$	$A_{3,3}$

MATRIX A

$B_{0,0}$	$B_{0,1}$	$B_{0,2}$	$B_{0,3}$
$B_{1,0}$	$B_{1,1}$	$B_{1,2}$	$B_{1,3}$
$B_{2,0}$	$B_{2,1}$	$B_{2,2}$	$B_{2,3}$
$B_{3,0}$	$B_{3,1}$	$B_{3,2}$	$B_{3,3}$

MATRIX B

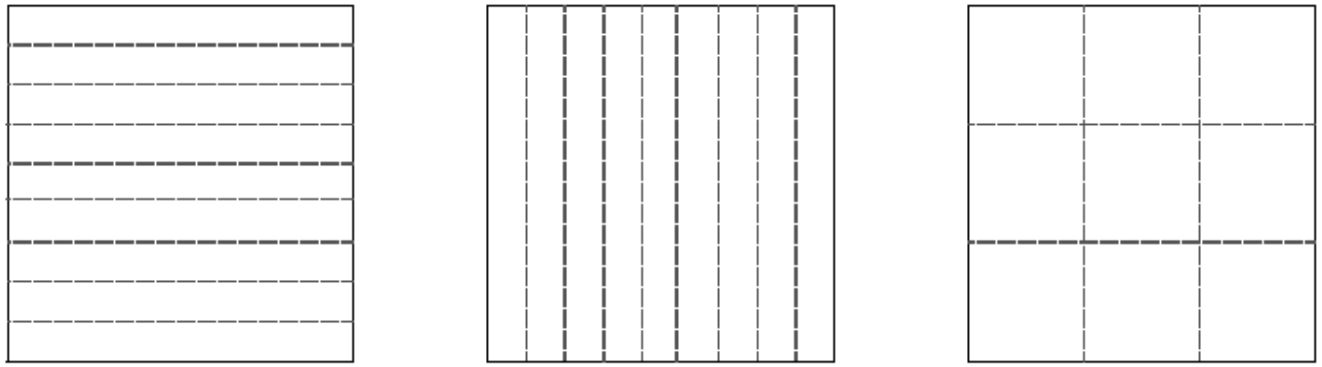
Resultant Matrix



Matrix A and Matrix B are Divided among 4 Processor P0 ,P1, P2, P3.

Partitioning & Mapping

Regular Partitioning



By Row By Column By Block

Different parallel formulations :-

row-wise striping, column-wise striping, or checkerboard striping.

Parallel Programming Paradigms

- On a single machine with shared memory
  - **Pthreads**
  - **OpenMP 3**
- On machines connected via network and have no shared memory
  - **MPI**
  - **PGAS**
- Accelerators, offload tasks from CPU to it
  - **CUDA**
  - **OpenACC /OpenMP 4**
  - **OpenCL**

Thank you

Questions ?