Parallelism

- Explicit - given by us
- Implicit - compiler does .

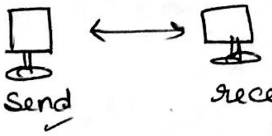
Executing simultaneously.

Methods

→ Message Passing
→ Data Parallel
→ Shared Memory
→ Remote Memory Operation
→ Threads
→ Combined Models .

Message Passing.

→ Each Processor has direct access to its local memory.
→ High speed.
→ Data exchange between processors are explicit
→ Co-operative operations



Prg

```
from mpi4py import MPI
comm = MPI.comm_WORLD  # gives access to processors
size = comm.Get_size()
rank = comm.Get_rank()
print ("Rank:", rank, "It Size", size)
```

```
mpiexec -n 4 python mes_pas.py.
```

O/P

```
Rank 2   Size 4
Rank 3   Size 4
```

Rank 0 size 4

Rank 1 Size 4

Data Parallel

- → All process work on diff segments of the same problem/data structure
- → Global/Local memory access for all process.
- → Concurrent access ~~must~~ must be co-ordinated.
- → message passing is done invisibily.

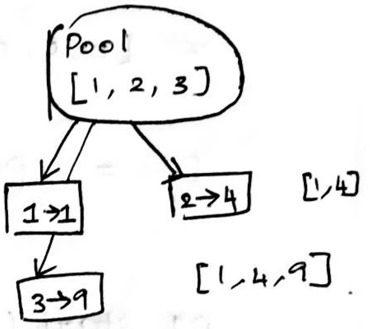
Prg:

```
from multiprocessing import Pool
import os

def f(x):
    return x * x

workers = os.cpu_count()

if __name__ == "__main__":
    with Pool(workers) as p:
        print (p.map(f, [1, 2, 3]))
```



Shared Memory

- → A set of process sharing common memory space.

Prg:

```
Import multiprocessing as mp

def sq(mylist , result ).
    for i, num in enumerate (mylist) :
        result [i] = num * num

if __name__ = "__main__" :
    mylist = [1, 2, 3, 4]
    result = mp. Array ('i', 4)
    # i → integer
    # 4 → size.
    # [0, 0, 0, 0] .

    p1 = mp. Process (target = sq ,
                       args (mylist , result ))

    p1 .start ()
    p1 .join()

    print (result [:]) .
#O/P [1 , 4 , 9, 16]
```

Remote Memory Operation

→ Set of process can access another process memory without its participation .

Same previous code .

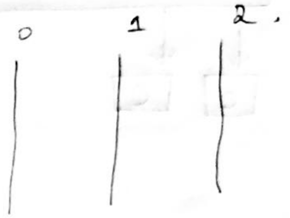
Threads

→ single process having multiple execution paths .

Prog :

```
import time
from threading import Thread

def sleeper (i) :
        print (i)
        time.sleep(2)
        print ("Done")
```



```
for i in range (3) :   # 0  1  2
        t = Thread (target = sleeper , arg =(i , ))
        t. start ( ) .
```

```
#O/P .

  0 1 2

  Done Done Done .
```

Ways . for Parallesim (

① Functional Decomposition , - small / fixed calculations .

② Domain Decomposition - Large / compulation is huge .

- Functional Parallelism
- Data Parallelism

Concepts .

①→ Phase parallel

②→ Divide & conquer

③→ Pipeline

④→ Process farm

⑤→ Work pool

Phase

- → computational phase
- → interaction phase .