Domain Decomposition Method – Parallel Computing

Somnath Roy
Department of Mechanical Engineering

Why Parallelization?

1. To reduce computational time by dividing the number of operations into a large number of computers

2. To reduce the matrix size to be stored in a single chunk of memory

Estimates of computational cost for large-scale problems

Grid spacing to resolve smallest turbulence length scale ~$(Re)^{-3/4}$

For Re~10^6, three-dimensional geometry needs 10^{13} grid points

One time-step (one set of matrix solutions) will need O(10^{13}) FLOP (Floating Point Operation)

Typical physical duration of a time step ~ 10^{-3} seconds

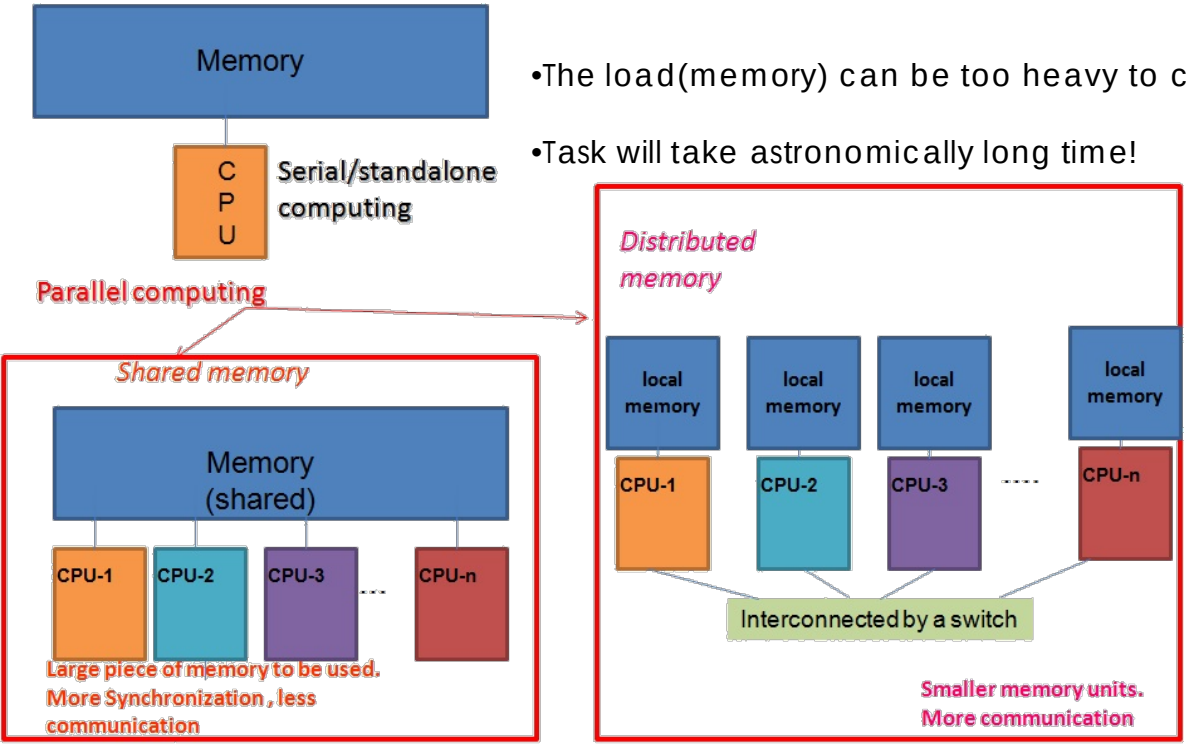
Simulation for 10 seconds will need 10^{17} FLOP

Fastest computers give 10 giga FLOP/sec speed

Estimated time for this calculation : 10^{10} seconds = 317 years !!!!

Matrix size = O(10^{13}) floats = 10^{13}x16 bytes >> standard RAM size

Parallel Computing- Basic Idea



- The load(memory) can be too heavy to carry
- Task will take astronomically long time!

distribution

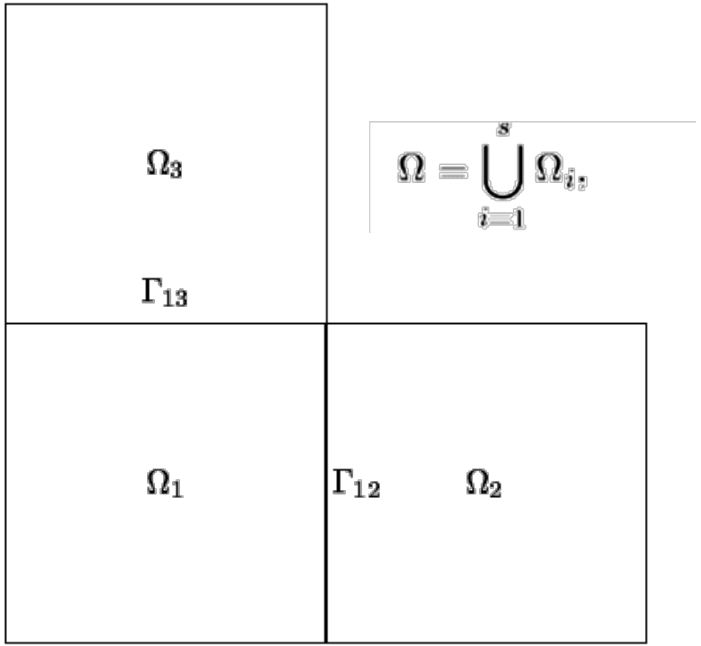
synchronization

communication

Tasks like Synchronization and communication is done by functions called using libraries as MPI, OpenMP

CUDA computing using graphics cards

Domain Decomposition



$$\Omega = \bigcup_{i=1}^{s} \Omega_i,$$

Distribute the domain into several subdomain

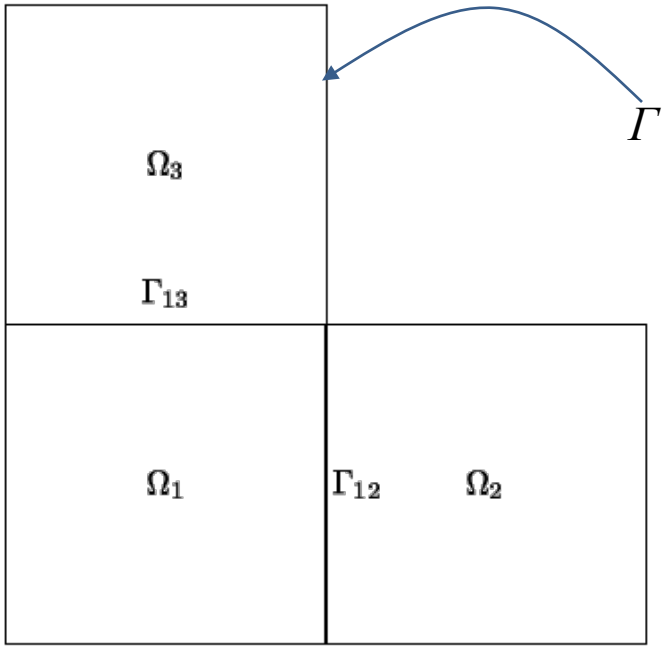
Form the matrix equation for each subdomain using the inter-boundary domain values

Propose an algorithm to solve each domain independently

Transfer inter-domain solutions to obtain continuity across the boundaries!

Converge to a final solution involving sundomains.

The matrix solution in decomposed domain



Solve: $\Delta u = f$ in $\Omega = \bigcup_{i=1}^{s} \Omega_i, \quad \Delta = \nabla^2$

With boundary conditions: $u = u_\Gamma \text{ on } \Gamma = \partial\Omega$

The matrix equation: $Au=b$

Let $x \equiv \Sigma x_i$ be the solution at the domain-internal points and y be the solution in the inter domain boundaries Γ_{ij}. Then $Au=b$ can be written as:

$$Au = b$$

$$\Rightarrow \begin{bmatrix} B_1 & & & E_1 \\ & B_2 & & E_2 \\ & & B_3 & E_3 \\ F_1 & F_2 & F_3 & C \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ y \end{Bmatrix} = \begin{Bmatrix} f_1 \\ f_2 \\ f_3 \\ g \end{Bmatrix}$$

The matrix solution in decomposed domain

$$Au = b$$

$$\Rightarrow \begin{bmatrix} B_1 & & & E_1 \\ & B_2 & & E_2 \\ & & B_3 & E_3 \\ F_1 & F_2 & F_3 & C \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ y \end{Bmatrix} = \begin{Bmatrix} f_1 \\ f_2 \\ f_3 \\ g \end{Bmatrix} \quad \Rightarrow \quad A\begin{Bmatrix} x \\ y \end{Bmatrix} = \begin{Bmatrix} f \\ g \end{Bmatrix}$$

$$\begin{bmatrix} B & E \\ F & C \end{bmatrix} \begin{Bmatrix} x \\ y \end{Bmatrix} = \begin{Bmatrix} f \\ g \end{Bmatrix}$$

or,

$$Bx + Ey = f \quad \text{and} \quad Fx + Cy = g$$

$$x = B^{-1}\left(f - Ey\right)$$

$$So, \quad FB^{-1}\left(f - Ey\right) + Cy = g$$

$$\underbrace{\left(C - FB^{-1}E\right)}_{S}\, y = g - FB^{-1}f$$

S (Schur Component)

$$\Rightarrow Sy = g - FB^{-1}f$$

$$\Rightarrow y = S^{-1}\left(g - FB^{-1}f\right)$$

So, finally the internal point solutions can be obtained as

$$x = B^{-1}\left(f - Ey\right)$$

with

$$y = S^{-1}\left(g - FB^{-1}f\right)$$

Schur Component

In a domain decomposition problem, Schur component is defined as

$$S = \left(C - FB^{-1}E\right)$$

Solution at the internal points of different subdomains are found as:

$$x = B^{-1}\left(f - Ey\right), y = S^{-1}\left(g - FB^{-1}f\right)$$

If S^{-1} exists, y can be found and hence x can also be found

Domain decomposition parallelization

To solve: $x = B^{-1}(f - Ey), y = S^{-1}\left(g - FB^{-1}f\right)$

B is a block diagonal matrix,

$$Au = b$$

$$\Rightarrow \begin{bmatrix} B_1 & & & E_1 \\ & B_2 & & E_2 \\ & & B_3 & E_3 \\ F_1 & F_2 & F_3 & C \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ y \end{Bmatrix} = \begin{Bmatrix} f_1 \\ f_2 \\ f_3 \\ g \end{Bmatrix}$$

B

Hence, inverse of B can be found in a decoupled sense as disjoint processes.

Hence the sets of equation given can be solved, provided y is made available to the particular process.

Schwarz Alternating procedure

Alternate between the domains for solution. Solve Dirichlet problem one one domain in each iteration and consider boundary conditions based on the most recent solution of the other domains.

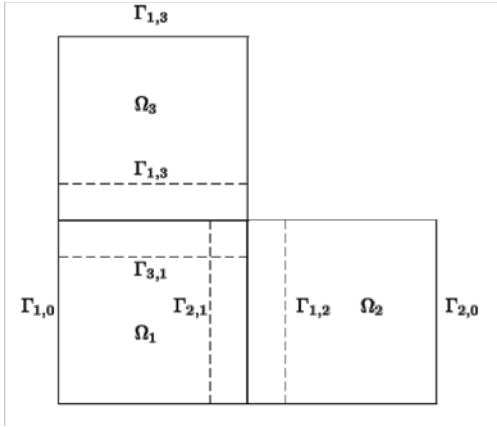
Algorithm

1. Choose an initial guess u to the solution
2. Until convergence Do:
3. For $i = 1, \cdots, s$ Do:
4. Solve $\Delta u = f$ in Ω_i with $u = u_{ij}$ in Γ_{ij}
5. Update u values on Γ_{ji}, $\forall j$
6. EndDo
7. EndDo

$$\Delta u = \nabla^2 u$$

- The algorithm sweeps through the s subdomains and solves the original equation in each domain based on the boundary conditions that are updated from the most recent values of u.
- We can start with a global initial guess and update it in each domain during the iterations

Schwaz Multiplicative Procedure for Overlapping Domains



Algorithm

1. *For* $i = 1, \cdots, s$ *Do:*
2. *Solve* $A_i \delta_i = r_i$
3. *Compute* $x_i := x_i + \delta_{x,i}$, $y_i := y_i + \delta_{y,i}$, *and set* $r_i := 0$
4. *For each* $j \in N_i$ *Compute* $r_{y,j} := r_{y,j} - E_{ji}\delta_{y,i}$
5. *EndDo*

Schwaz Multiplicative Procedure - Steps

1. Chose an initial guess u to the solutions
2. Iterate until convergence
3. For $i=1,\ldots.s$
4. Solve $\Delta u=f$ in Ω_i with $u=u_{ij}$ in Γ_{ij}
5. Update u values in Γ_{ij}
6. Till convergence in all $\Omega=\Omega_1,\Omega_2,\ldots. \Omega_s$

Convergence?

Theorem for Convergence of Schwarz Procedure

If the guess $\begin{pmatrix} x_i^{(0)} \\ y_i^{(0)} \end{pmatrix} = u_i^{(0)}$ is chosen as $x_i^{(0)} = B_i^{-1}\left[f_i - E_i y_i^{(0)} \right]$ then the iterations are identical to Gauss-Seidel sweep of the Schur component and they converge!

Domain Decomposition based parallel Matrix Solver

1. Divide the domain into a number of subdomains. Domain overlaps are allowed such that the full row equation for each internal point of the subdomain is available
2. Start with a global guess
3. Update the solution at every subdomain locally. Consider the inter-domain boundaries as Dirichlet with the last updated solution value **– parallel step**
4. Update the boundary values in one sub-domain as obtained by local solution of neighbouring domains. **– Data transfer step**
5. Iterate over the domains for a global convergence **–synchronization step**

Parallel Computing: Basic Idea

Serial/standalone computing

- The load(memory) can be too heavy to carry
- Task will take astronomically long time!

distribution

synchronization

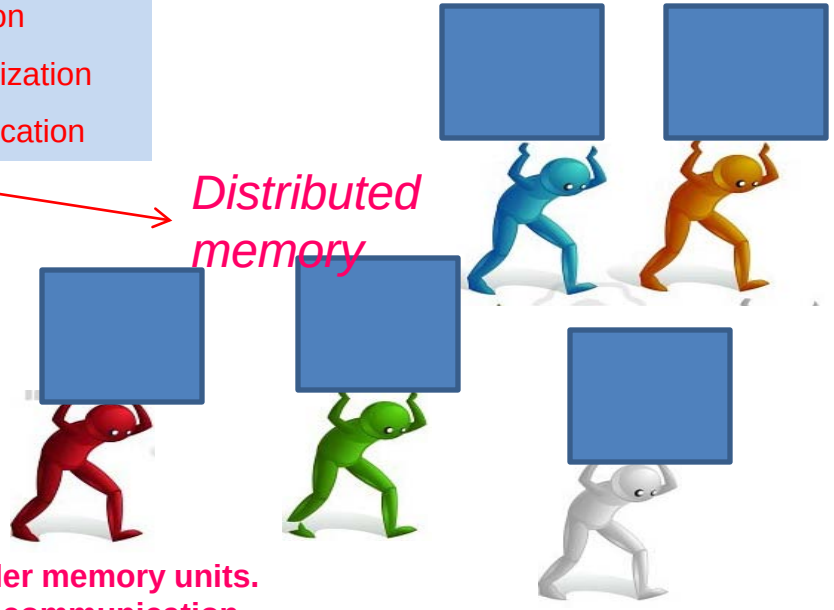
communication

Parallel computing

Shared memory (load)

Large piece of memory (or load) to be used. More Synchronization , less communication

Distributed memory



Smaller memory units. More communication

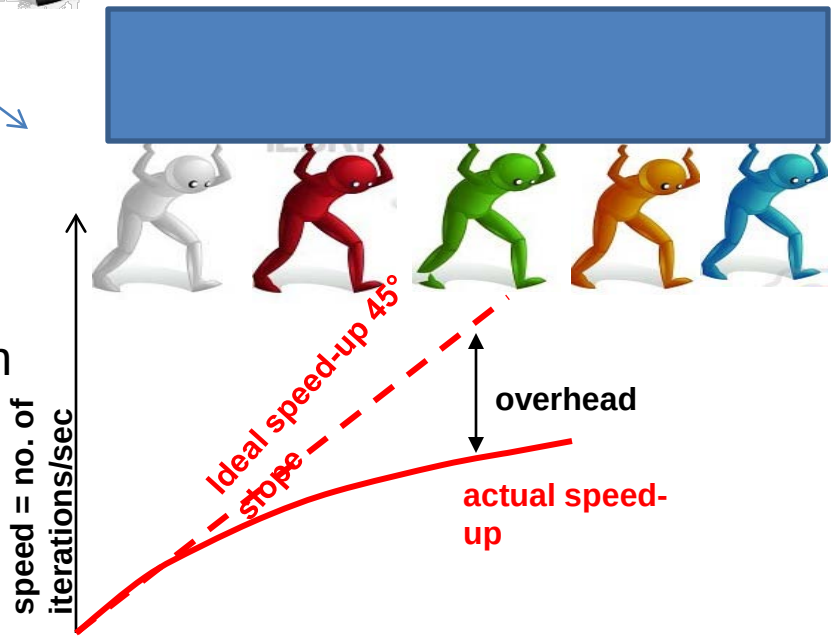
Parallel Computing: The More the Better?

Less weight for each people

--Increase in speed

- Non-uniform distribution
- More communication and synchronization
- Idle time for some of them

--More latency



Architecture of a Parallel Computing Platform

Architecture of a supercomputer (Flynn's taxonomy):

IBM Blue gene (Aregon national lab)

- **250000 processors (2GHz)**
- **10-Gigabit ethernet connector**
- **petaflops speed, petabyte RAM**

Based upon the number of concurrent instruction (or control) and data streams available in the architecture (Flynn, 1966):

Single Instruction, Single Data stream (SISD):

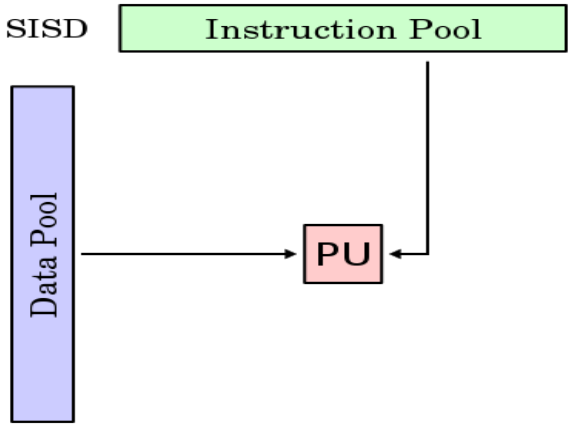
Single Instruction, Multiple Data streams (SIMD)

Multiple Instruction, Single Data stream (MISD)

Multiple Instruction, Multiple Data streams (MIMD)

SISD

- Computers in this category can decode only a single instruction in unit time



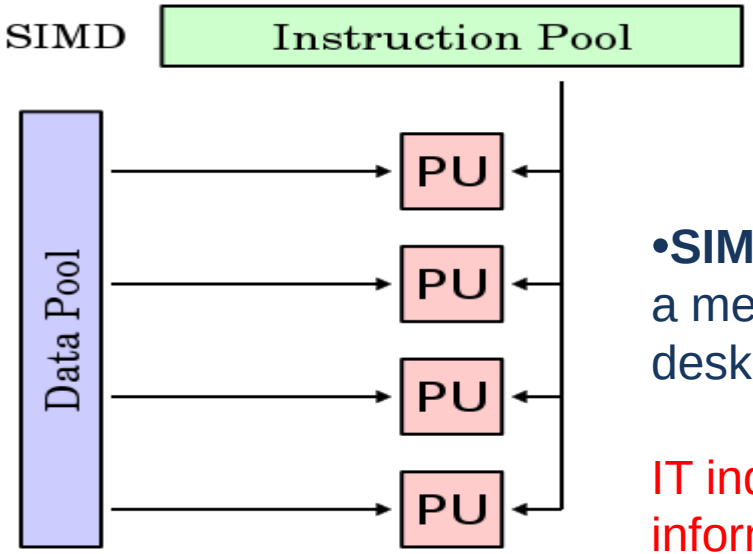
An analogy of Flynn's classification is **the check-in desk at an airport**

- **SISD**: a single desk

The way an ordinary computer works

SIMD

- An array of processors all executing the same instruction but on different data

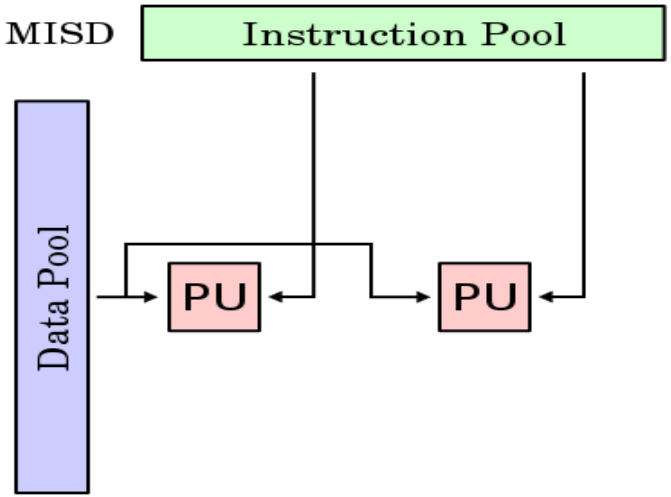


•**SIMD**: many desks and a supervisor with a megaphone giving instructions that every desk obeys

IT industry jobs: processing of credit card information for 1 lac people in 1000 computers

MISD

- Some consider this category to be empty. However, some consider **systolic arrays** to fall in this category:
  - Data is pumped through processors, each processor applying a different operation to the same data stream.

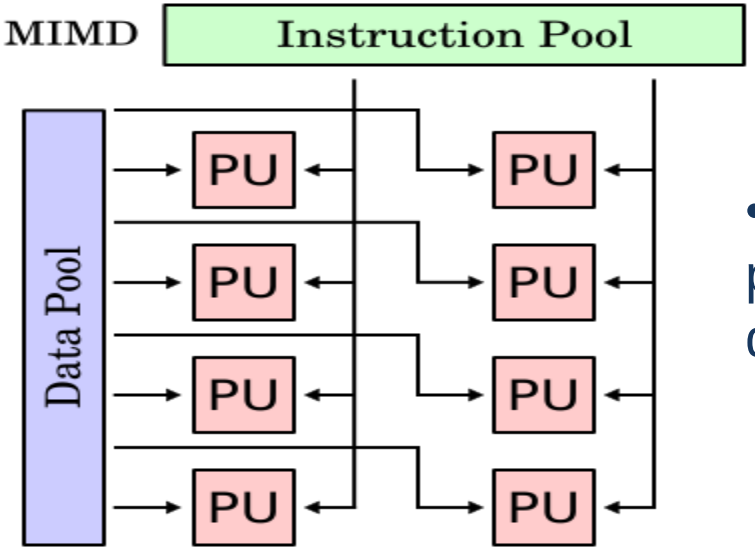


•**MISD**: For the same passenger different desks are doing different job. One processing ticket, the other checking luggage etc.

Income Tax Department:
Processing different financial information (Tax, Bank transaction, Foreign money exchange) from a same PAN

MIMD

- MIMD: More than one CPU, each running its own program on its own data



•**MIMD**: many desks working at their own pace, synchronized through a central database

More communication between each processor

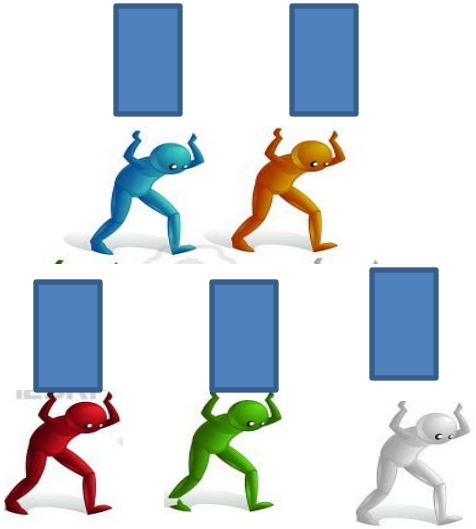
More flexibility

Used in scientific computing

Single Program Multiple Data & Multiple Program Single Data models

Elements of a Parallel Program

What we need to know



How many people doing the work. (Degree of Parallelism)

What is needed to begin the work. (Initialization)

Who does what. (Work distribution)

Access to work part. (Data/IO access)

Whether they need info from each other to finish their own job. (Communication)

When are they all done. (Synchronization)

What needs to be done to collate the result.

Message Passing Interface (MPI)

What is MPI?

- A message passing library specification
  - Message-passing model
  - Library functions can be used with C, Fortran, C++ compilers

  In short:
  - MPI "is a message-passing application programmer interface, together with protocol and semantic specifications for how its features must behave in any implementation.“

- Designed for parallel computers, clusters, and heterogeneous networks

The other popular interface available is OpenMP

Why MPI?

- Small
  - Many programs can be written with only 6 basic functions

eg. ‘call MPI_SEND(start, count, datatype, dest, tag, comm,ierr)’

- Large
  - MPI’s extensive functionality from many functions
- Scalable
  - Point-to-point communication
- Flexible
  - Don’t need to rewrite parallel programs across platforms

And a Free Open-source Software!

MPI Functions

Many parallel programs can be written using just these functions,
and they are:-

MPI_INIT	**Initialize MPI**
MPI_FINALIZE	**Exit MPI**
MPI_COMM_SIZE	**Determine number of processes within a comm**
MPI_COMM_RANK	**Determine process rank within a comm**
MPI_SEND	**Send a message**
MPI_RECV	**Receive a message**
MPI_COMM_WORLD	**Default communicator whose group contains all initial processes**

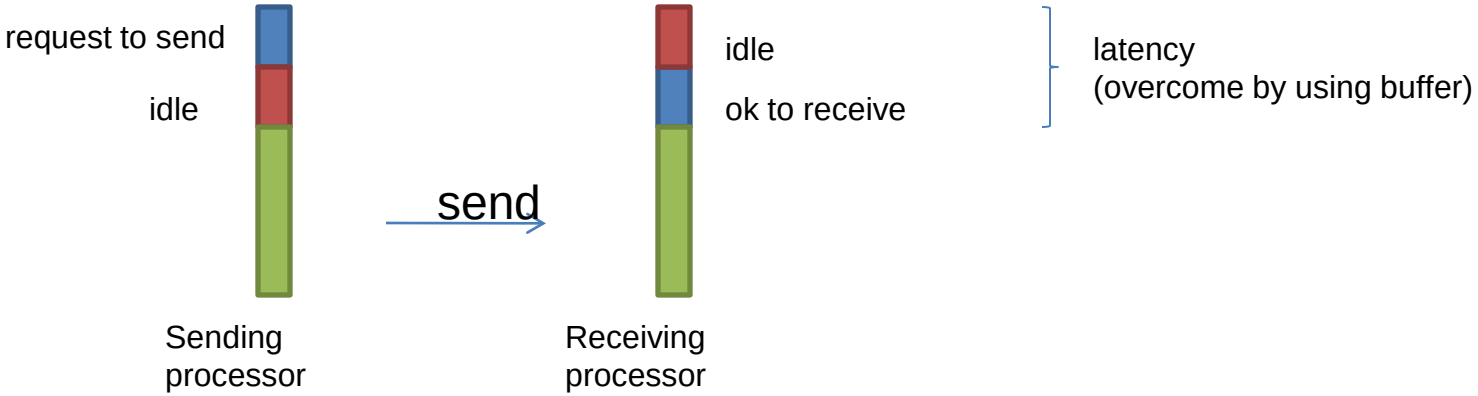
Point to Point Communication

MPI_SEND(start, count, datatype, dest, tag, comm,ierr)

- call MPI_SEND(tsendr,ndata,MPI_DOUBLE_PRECISION,myid+1,10, MPI_COMM_WORLD,ierr)

!MPI_RECV(start, count, datatype, source, tag, comm,status,ierr)

- call MPI_RECV(trecvl,ndata,MPI_DOUBLE_PRECISION,myid-1,10, MPI_COMM_WORLD,stt, ierr)

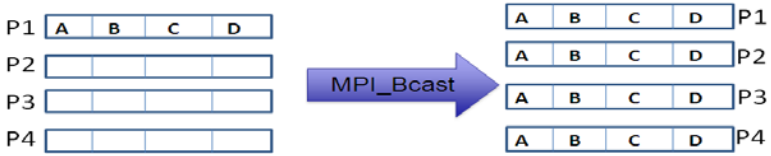


Note* : Other than the above standard send and recv , the others blocking send or recv are **: MPI_Ssend , MPI_Bsend ,MPI_Srecv , MPI_Brecv**

Collective Communication

- Collective functions involve communication among all processes in a process group, Instead of involving communication between two specific processes. **i.e.** A single call handles the communication between all the processes in a communicator

- **MPI_BCAST** distributes data from one process (the root) to all others in a communicator.

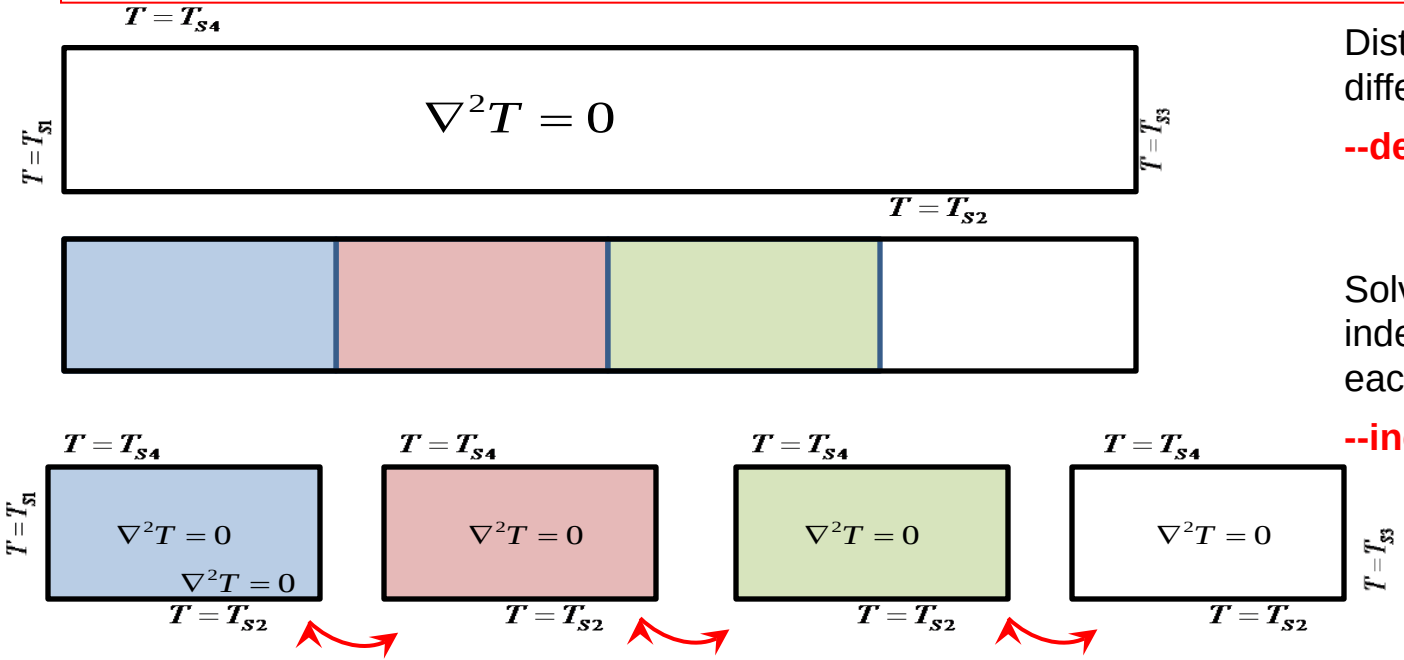


- **MPI_REDUCE** combines data from all processes in communicator and returns it to one process.

Note*: In many numerical algorithms, SEND/RECEIVE can be replaced by BCAST/REDUCE, improving both simplicity and efficiency.

Parallelization of CFD Problem : Domain Decomposition

Partitioning the domain into smaller blocks and solving smaller matrices in different computers in parallel



Distributing the domains to different computer

--decreases matrix size

Solving each domain independently and parallelly in each computer

--increases speed

Dynamic exchange of boundary condition among each domain over a partition line

Key Points in Parlallel Program

✓ Initialization of Parallel environment

✓Allocation of decomposed domain to the processors

-- load balancing, idle time minimization

✓Calculation in each domain

✓Synchronization

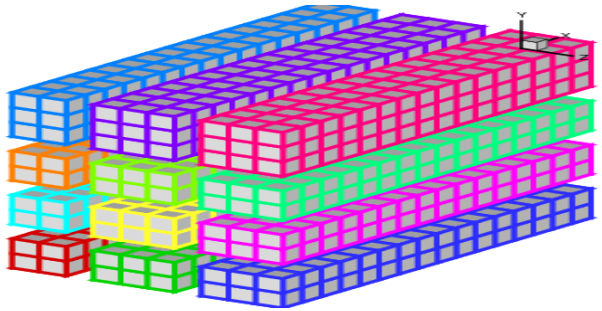
-- latency

✓Communication

-- avoiding bottleneck or deadlock in data exchange!

✓Assembly of results

✓Termination



overheads due to initialization, synchronization and communication

Scalability of an MPI Program

Geometry with 3 million grid points
Block structured mesh. Each block is solved in a separate processor
Data exchange for two layers among each partition