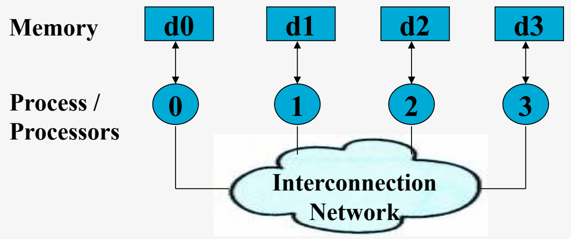
Chapter 3. Communication between tasks

- Parallel computing
  - Dependent tasks executed by different process/threads
  - Need of communication between tasks
- Communication way
  - Shared memory
    - Transparency for programmer
    - Memory copy limited
    - Concurrent access control: semaphores
    - Programming: thread, openMP
    - Data coherence if distributed memory
    - Disadvantages: lack of scalability (memory & nb. of CPUs)

Communication way

- Communication way
  - Message passing scheme
    - Use of *send()* and *recv()*



Communication way

- Communication way
  - Message passing scheme
    - Communication mode:
      - Blocking / Non-blocking communication

        Blocking: send() finish when recv() is performed.
      - Synchronous / Asynchronous communication

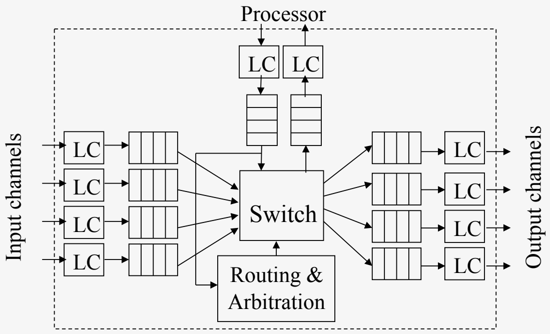
        Synchronous: “handshaking” between tasks before communication.
      - Blocking ←→ Synchronous
      - Asynchronous ←→ Non-blocking

Communication Layers

- Interprocessors communication layers :
  - *Physical layer* : link-level protocols for transferring message and managing the physical channels between adjacent routers.
  - *Switching layer* : implements mechanisms for forwarding message through the network.
  - *Routing layer* : makes routing decisions.

Generic Router Model

- *LC*: Link Controllers



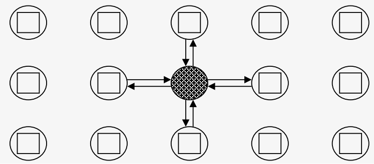
Communication --- Example

❑ Local communication

- Jacobi finite difference

$$X_{i,j}^{(t+1)} = \frac{4X_{i,j}^{(t)} + X_{i-1,j}^{(t)} + X_{i+1,j}^{(t)} + X_{i,j-1}^{(t)} + X_{i,j+1}^{(t)}}{8}$$

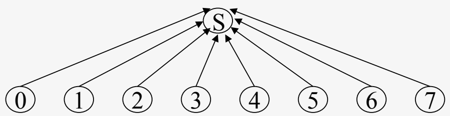
- Each task compute X , X , X , ...



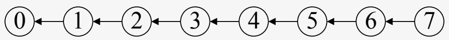
Communication --- Example

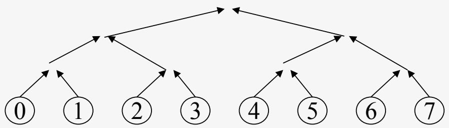
❑ Global communication: $S = \sum_{i=0}^{N-1} Xi$

- Centralized summation



- Computing techniques





Communication

❑ Unstructured and dynamic Communication

- In Master / Slaves programming model

❑ Asynchronous communication

- Producers are not able to determine when consumers may require data
- Consumers must explicitly request data from producers
- Computation tasks / Data tasks

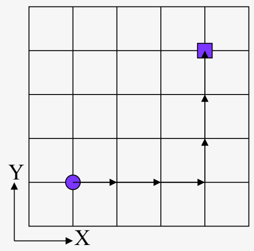
Communication Type

- ❑ Point-to-point / One-to-one
- ❑ Collective communication
  - One-to-all
    - Broadcast, Scatter
  - All-to-one
    - Gather, Reduce
  - All-to-all
    - All_broadcast (Gossiping or total exchange)
    - All_scatter (Complete exchange)
  - Barrier synchronization

Point-to-Point Communication

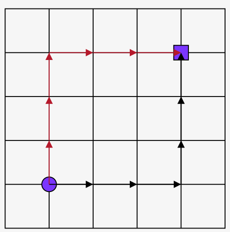
- ❑ XY routing for 2D-meshes
  - Inputs: coordinates of current node (X_c, Y_c) and destination node (X_d, Y_d)
  - Outputs: selected output channel

Procedure :
$X_{offset} = X_d - X_c$;
$Y_{offset} = Y_d - Y_c$;
if $X_{offset} < 0$ then Channel = X-;
if $X_{offset} > 0$ then Channel = X+;
if $X_{offset} = 0$ and $Y_{offset} < 0$ then
Channel = Y-;
if $X_{offset} = 0$ and $Y_{offset} > 0$ then
Channel = Y+;
if $X_{offset} = 0$ and $Y_{offset} = 0$ then
Channel = Internal;

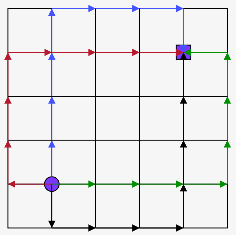


Point-to-Point Communication

- ❑ XY routing for 2D-meshes



2-minimal paths

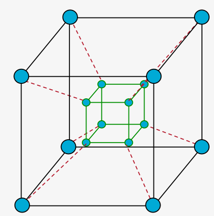


4-paths : d = d(S,D)+2

Point-to-Point Communication

- ❑ Dimension-order routing for hypercube
  - Inputs: addresses of current node *Current* and destination node *Dest*
  - Outputs: selected output channel
  - FirstOne(): returns the first bit set to one.

Procedure :
offset = Current - Dest;
if offset = 0 then
Channel = Internal;
else
Channel = FirstOne(offset);



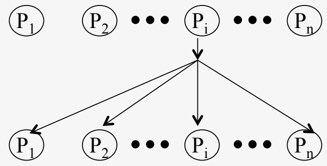
Point-to-Point Communication

- Dimension-order routing for hypercube: Example
  - Source: $S = X_{d-1} \ldots X_{i+1} X_i s_{i-1} \ldots s_1 s_0$
  - Destination: $D = X_{d-1} \ldots X_{i+1} X_i d_{i-1} \ldots d_1 d_0$
  - Path:

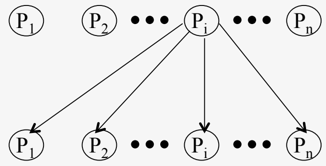
$$P_0 = S = X_{d-1} \ldots X_{i+1} X_i s_{i-1} \ldots s_1 s_0$$
$$P_1 = X_{d-1} \ldots X_{i+1} X_i s_{i-1} \ldots s_1 d_0$$
$$P_2 = X_{d-1} \ldots X_{i+1} X_i s_{i-1} \ldots d_1 d_0$$
$$\ldots$$
$$P_i = D = X_{d-1} \ldots X_{i+1} X_i d_{i-1} \ldots d_1 d_0$$

Collective Communication

- Process group
  - n process $P_1, P_2, \ldots P_n$
- One-to-all communication



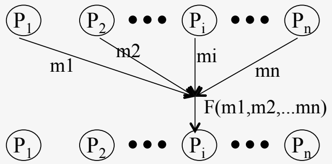
(a) Broadcast communication



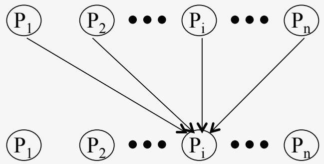
(b) Scatter communication

Collective Communication

- All-to-one communication



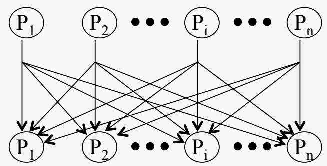
(c) Reduce communication



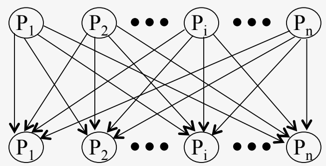
(d) Gather communication

Collective Communication

- All-to-all communication



(e) All-broadcast communication

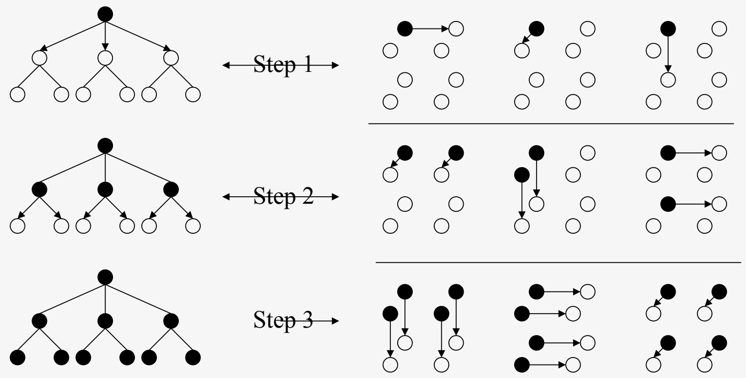


(d) All-scatter communication

- Barrier synchronization
  - A logical point of synchronization for all process in a group
  - A reduce followed by a broadcast operation

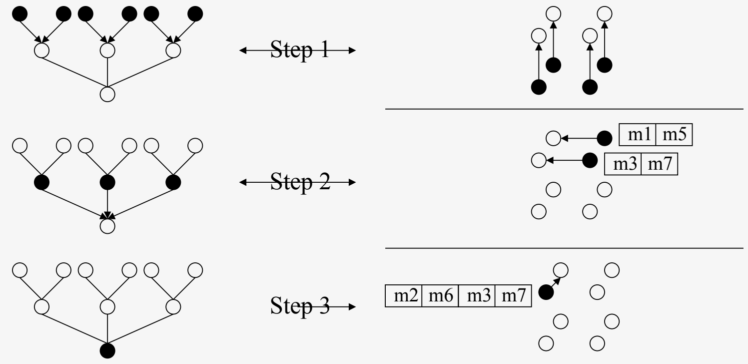
One-to-all --- Algorithms

- Tree
- Hypercube



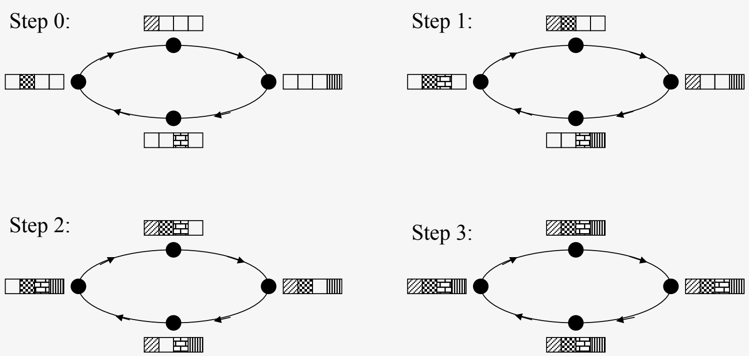
All-to-one --- Algorithms

- Tree
- Hypercube



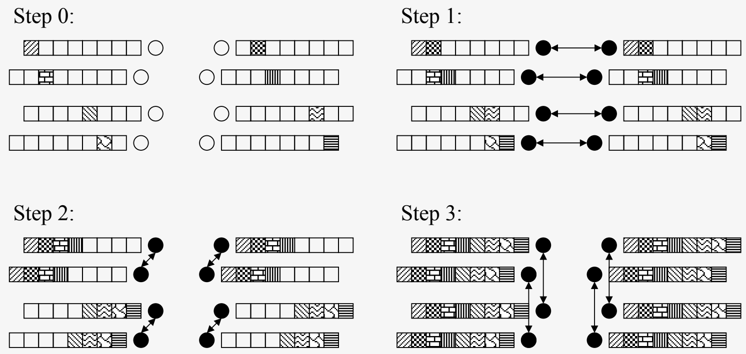
All-to-all --- Algorithms

- Undirected ring



All-to-all --- Algorithms

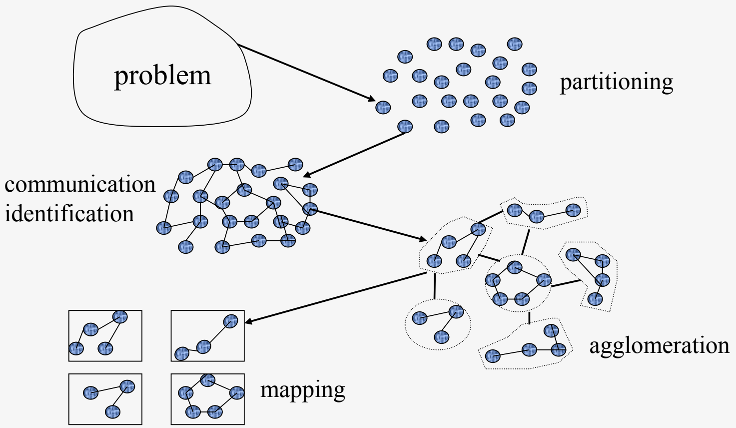
- Hypercube



Communication --- General

- ❑ Objective
  - Determine the dependence between the tasks
  - Determine the communications requirements
- ❑ Communication categories
  - Local/Global communication
  - Structured/Unstructured communication
    - Form of communication structure
  - Static/Dynamic communication
    - Change of communication partners over time
  - Synchronous/Asynchronous communication

Chapter 4. Design of parallel programs

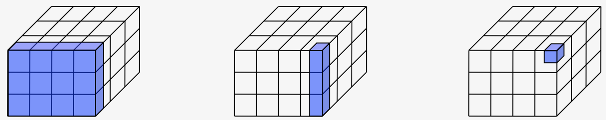


Partitioning --- general

- Objective
  - Decomposition of the computation of a problem and data into small tasks
  - Avoidance of replicating computation and data
  - Balance of work between tasks
- Partitioning techniques
  - *Domain decomposition*: determine first an partition of the data, then work out how to associate computation with data
  - *Functional decomposition*: decomposing first the computation, then dealing with data

Partitioning --- domain decomposition

- Operations
  1. Decomposing the data into small pieces of ~ equal size
  2. Partition of the computation associated to data
  - Different phases of the computation operate on different data structures ⇒ Treat each phase separately
- Examples

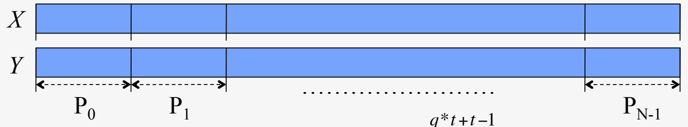


Dot Product

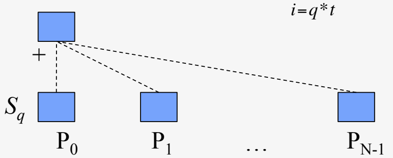
- Problem
  - Computation of $S = \sum_{i=0}^{n-1} X_i * Y_i$
- Assumptions
  - *n:* big number
  - *N* process
  - Only one process have initial data: *X*, *Y* and *n*

Dot Product

- Parallel algorithm
  - Distribution of the components of X and Y



  - Computation of local S_q: $S_q = \sum_{i=q*t}^{q*t+t-1} X_i * Y_i,\ \ t = \frac{n}{N}$
  - Reduce:

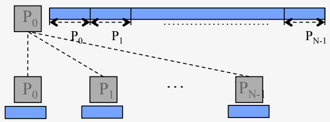


Dot Product

- Distribution

```
…
#define ARRAYSIZE 2048
…
int myrank, nbprocs, blocsize;
double *X=NULL, *MX=NULL;
MPI_Comm_rank( MPI_COMM_WORLD, &myrank );
MPI_Comm_size( MPI_COMM_WORLD, &nbprocs);
blocsize=ARRAYSIZE/nbprocs;
if (myrank==0){
  X=(double *)malloc(ARRAYSIZE*sizeof(double));
  for (i=0; i<ARRAYSIZE; i++)X[i]=i*i;
  for (i=1; i<nbprocs; i++)
    MPI_Send(X+i*blocsize, blocsize,
      MPI_DOUBLE, i, tag, MPI_COMM_WORLD);
}
else {
  X=(double *)
    malloc(blocsize*sizeof(double));
  MPI_Recv(X, blocsize, MPI_DOUBLE, 0,
    tag, MPI_COMM_WORLD, &status);
}
```

greedy

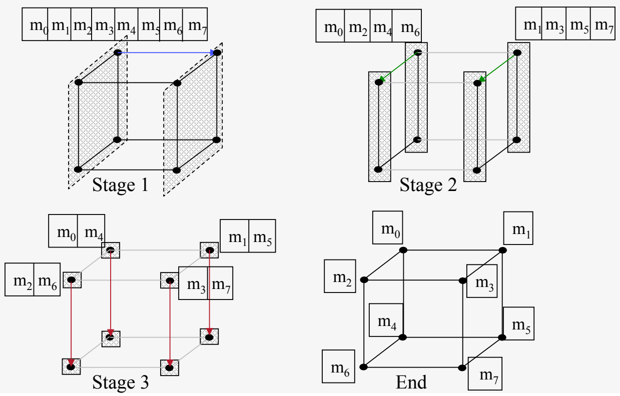


collective

```
if (myrank==0){
   X=(double *)
      malloc(ARRAYSIZE*sizeof(double));
   for (i=0; i<ARRAYSIZE; i++)X[i]=i*i;
}
MX=(double *)
    malloc(blocsize*sizeof(double));
MPI_Scatter(X, blocsize, MPI_DOUBLE,
            MX, MPI_DOUBLE, 0,
            MPI_COMM_WORLD);
```


Dot Product

- Distribution on a hypercube of degree *3*

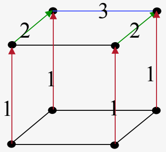


Dot Product

- Distribution on a hypercube of degree *d*
  - First stage
    - Cut the *d*-cube into two (*d-1*)-cubes, by suppressing the links of dimension *0*
    - P_0 send the data for process of 2nd (*d-1*)-cube to its neighbor in 2nd (*d-1*)-cube
  - i^{th} stage (i=*2, …, d*)
    - Repeat the operations of first stage on each (*d-i+1*)-cube; divide each (*d-i+1*)-cube by suppressing the links of dimension *i-1*

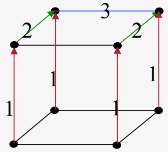
Dot Product

- Reduce on a hypercube of degree *d*
  - First stage
    - Cut the *d*-cube into two (*d-1*)-cubes, by suppressing the links of dimension *d-1*
    - Each processor of the 2^{nd} (*d-1*)-cube send its S_q to its neighbor in 2^{nd} (*d-1*)-cube, which adds with local S_q
  - i^{th} stage (i=*2, …, d*)
    - Repeat the operations of first stage on each (*d-i+1*)-cube; division of each (*d-i+1*)-cube by suppressing the links of dimension *d-i*



Dot Product

- Reduce on a hypercube of degree *d*




```
cont=1;
for (i=d-1; cont && i>=0; i--) {
   biti = ((1<<i)&rang) >> i;  // What I do (1-send/0-recv)
   neigbi= (1<<i)^rang; // With who I communicate
   if (biti) {
      MPI_Send(&res, 1, MPI_INT, neigbi,
                  tag, MPI_COMM_WORLD);
      cont= 0;//exit after send
   }
   else {
      MPI_Recv(&tmp, 1, MPI_INT, neigbi,
                  tag, MPI_COMM_WORLD, &state);
      res += tmp;
   }
}
```

```
/* Ordre d'un hypercube */
int bit_last1(int n)
{
    int d=0;

    while (n>>=1) d++;

    return d;
}
```

```
MPI_Reduce(&res, &res_g, 1, MPI_DOUBLE,
              MPI_SUM, 0, MPI_COMM_WORLD );
```


Dot Product

- Performance evaluation
  - Time for computation:

$$Tcomp_q = t * t_{mult} + (t-1) * t_{add},\quad t = \frac{n}{N}$$

  - Time for communication: Model: $T_{com}(L) = \beta + L\tau$

$$Tdist = d\beta + \left[2^{d-1} + \ldots + 2^1 + 2^0\right] * t * sizeof(component)\tau$$
$$= d\beta + t(2^d - 1) sizeof(component)\tau$$
$$Tred = d\beta + d * sizeof(component)\tau + dt_{add}$$

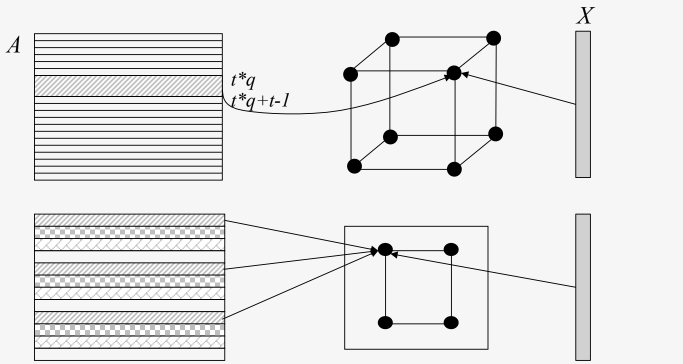
  - Total execution time: $Ttotal = Tcomp + Tdist + Tred$

Matrix-Vector Multiplication

- Problem
  - Compute $V=AX$, A: matrix $n{\times}n$, and $X,V \in \Re^n$, n big number
- Parallel algorithm: line-version
  - Aim: each process compute several components of V
  - Algorithm
    1. Distribution of the lines of A and broadcast of X to each process
    2. Every process computes in parallel its V_i
    3. Gathering of the elements of V
    4. Possible broadcasting of V

Matrix-Vector Multiplication

- Partitionning: line-version $t = \frac{n}{N}$, q process number

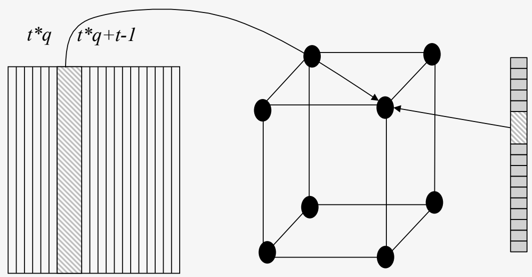


Matrix-Vector Multiplication

- Parallel algorithm: column-version
  - Aim: each process compute a partial value of all components of V
  - Algorithm
    1. Distribution of the columns of A and the components of X
    2. Every process compute simultaneously a partial value of all V_i
    3. Reduce of the components of V (with addition)

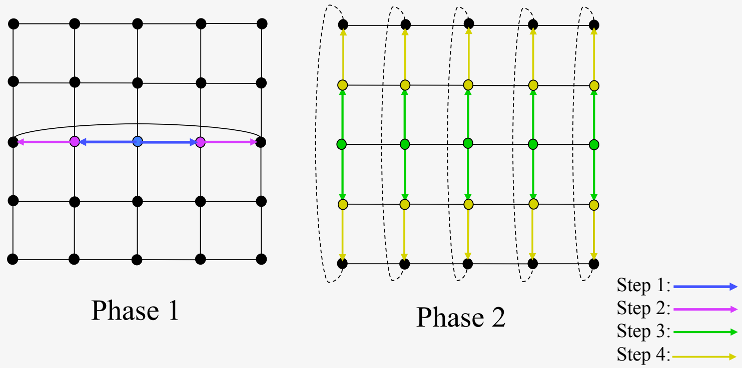
Matrix-Vector Multiplication

- Partitionning: column-version



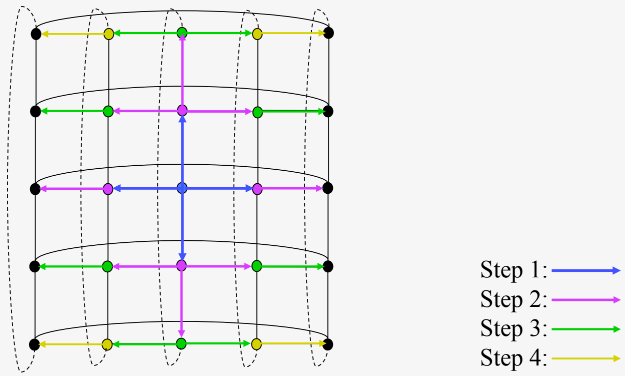
Matrix-Vector Multiplication

- Distribution – algorithm 1:



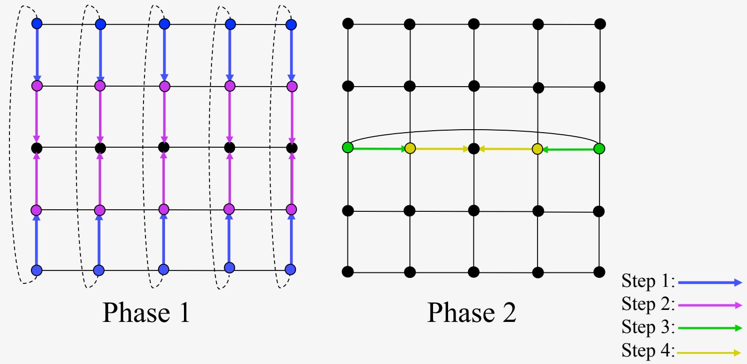
Matrix-Vector Multiplication

- Distribution – algorithm 2:



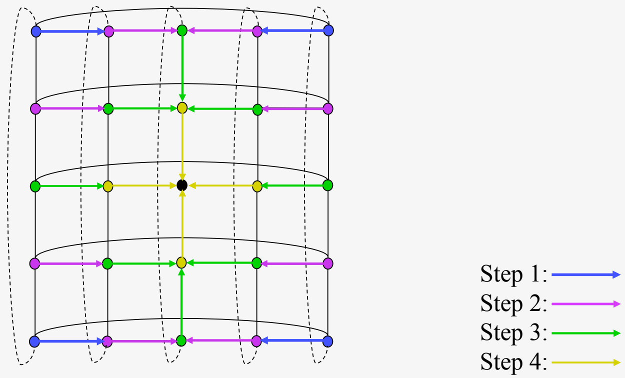
Matrix-Vector Multiplication

- Gathering / reduce – algorithm 1:



Matrix-Vector Multiplication

- Gathering / reduce – algorithm 2:

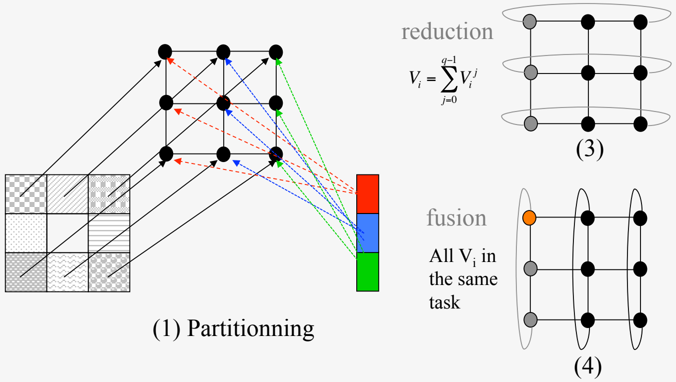


Matrix-Vector Multiplication

- Parallel algorithm: bloc-version on a torus
  - Aim: each process compute a partial value of a equal number of V_i
  - Algorithm
    1. Distribution of the blocs of A and the elements of X
    2. Every process computes in parallel a partial value of its V_i
    3. Horizontal all-to-all for computing total value of V_i
    4. Vertical all-to-all for communicating all V_i to all process

Matrix-Vector Multiplication

- Parallel algorithm: bloc-version on a torus



Matrix-Vector Multiplication

- Performance evaluation
  - Computation time
    - Line-version: computation of t components of V

$$Tcomp_{i,j} = t * \left[n * t_{mult} + (n-1) * t_{add} \right]$$

    - Column-version: computation of n partial values

$$Tcomp_{i,j} = n * \left[t * t_{mult} + (t-1) * t_{add} \right]$$

$$+ \text{ computation in reduce } = n * \left[\frac{p_x}{2} + \frac{p_y}{2} \right] * t_{add} \text{ , } p_x * p_y = N$$

Matrix-Vector Multiplication

- Communication time - line-version:
  - Distribution of t lines of A per process

$$Tdist_A^1 = \frac{p_x}{2} * \beta + \left[\frac{p_x}{2} + \ldots + 1 \right] * p_y * t * n * sizeof(component)\tau$$

$$= \frac{p_x}{2} * \left[\beta + \frac{\frac{px}{2}+1}{2} * p_y * t * n * sizeof(component)\tau \right]$$

$$Tdist_A^2 = \frac{p_y}{2} * \beta + \left[\frac{p_y}{2} + \ldots + 1 \right] * t * n * sizeof(component)\tau$$

$$= \frac{p_y}{2} * \left[\beta + \frac{\frac{py}{2}+1}{2} * t * n * sizeof(component)\tau \right]$$

Matrix-Vector Multiplication

- Communication time
  - Broadcast X to all process

$$Tbroad_X^1 = \frac{p_x}{2} * \left[\beta + n * sizeof(component)\tau \right]$$

$$Tbroad_X^2 = \frac{p_y}{2} * \left[\beta + n * sizeof(component)\tau \right]$$

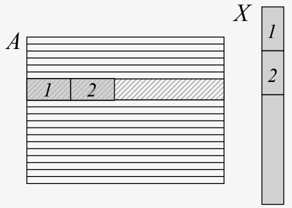
  - Gathering of V on one process (t components per proc)

$$Tgather_V^1 = \frac{p_y}{2} * \left[\beta + \frac{\frac{py}{2}+1}{2} * t * sizeof(component)\tau \right]$$

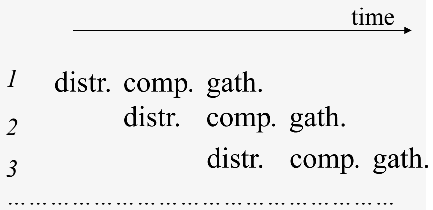
$$Tgather_X^2 = \frac{p_x}{2} * \left[\beta + \frac{\frac{px}{2}+1}{2} * p_y * t * sizeof(component)\tau \right]$$

Matrix-Vector Multiplication

- Improvement – line-version
  - Aim: overlap computation and communication
  - Previous parallelization: complete separation of 2 works



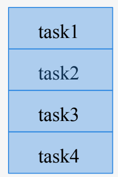
Partitionning



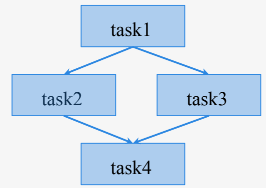
Tasks scheduling for blocs

Partitioning --- Functional Decomposition

- Functional Decomposition
  1. Dividing the computation into disjoint tasks
  2. Examining the data requirements of these tasks
  - Overlap of the data partition $\Rightarrow$ Use of communication to avoid the replication of data



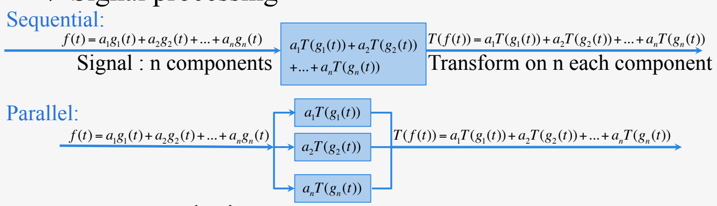
Sequential program



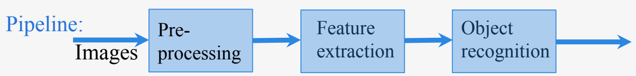
Parallel program

Partitioning --- Functional Decomposition

- Examples
  - Signal processing



  - Image analysis

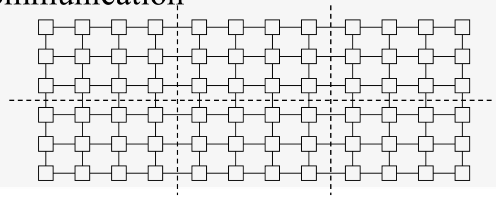


Mapping --- General

- Goals
  - Minimize total execution time
- Techniques
  - Place concurrent tasks on different processors
  - Place communicating tasks on the same processor
- Observations
  - NP-complete problem
  - Specialized strategies and heuristics for classes of problem

Mapping

- ❑ Classification of problems and associated techniques
  - Domain decomposition ⇒ fixed number of equal-size tasks and structured local and global communication ⇒ Minimize inter-processor communication



Mapping

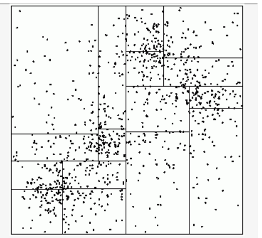
- ❑ Classification of problems and associated techniques
  - Domain decomposition generating variable-size tasks and/or unstructured communication patterns ⇒ Load balancing algorithms
  - The number of tasks, or the amount of computation and communication changes dynamically during program execution ⇒ Dynamic load balancing algorithms

Mapping --- Load Balancing

- ❑ Goals
  - One coarse-grained task per processor
  - Partitioning the computational domain to yield one sub-domain for each processor
- ❑ Techniques
  - *Recursive bisection*
  - *Cyclic mappings*
  - *Probabilistic methods*
  - *Dynamic work assignment*

Load balancing --- some techniques

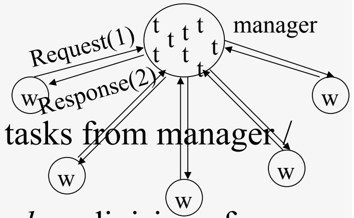
- ❑ Recursive bisection
  - Orthogonal division in alternating the x and y direction,
  - Equal number of items in each sub-domains



- ❑ Cyclic mapping
  - Mapping the tasks to process in cycle

0	1	2	3	0	1	2	3
0	1	2	3	0	1	2	3

Mapping --- dynamic work assignment



- Strategies
  - *Manager/worker*
    - Idle workers request tasks from manager / others workers.
  - *Hierarchical manager/worker*: division of workers into disjoint sets, each with a sub-manager
- Termination detection
  - Determining when idle works stop requesting.

Mapping --- dynamic work assignment

- Master / slaves – Y=AX

Master :

- Initialisation de A et X
- Diffusion de X
- Envoi d'une ligne de A à chaque esclave
- Si nb. de lignes de A < nb. d'esclaves
  - Terminer les esclaves en trop
- Boucle sur la réception des résultats et distribution des lignes de A restantes
  - Récupération d'un Yi
  - Envoi d'une nouvelle ligne de A s'il y en a

Slaves :

- Diffusion de X
- Réception d'une ligne de A
- Boucle sur
  - Calcul d'un Yi
  - Envoi d'un Yi au master
  - Réception d'une nouvelle ligne de A s'il y en a
  - Terminaison sinon

Keys of performance --- Synthèse

- Reduce the copies of data
- Load balancing
- Use of non blocking communication
- Reduce the synchronization
- Recovery of the communication by the calculation
- Use dynamic work assignment