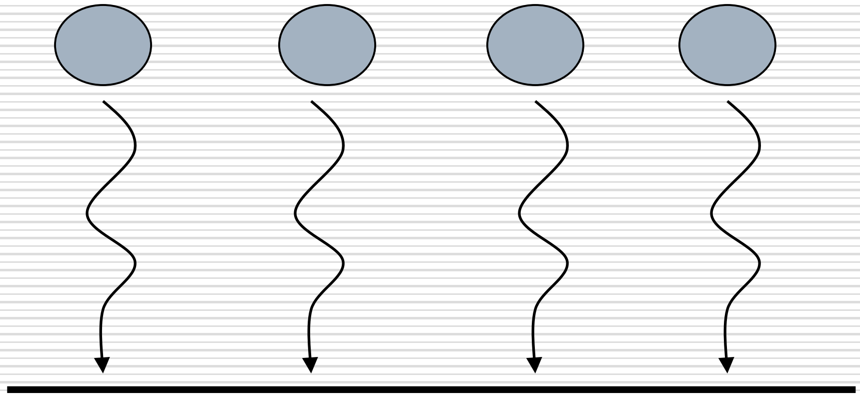
Parallel Algorithms

Sathish Vadhiyar

COLLECTIVE ALGORITHMS

Collective Communications - Barrier

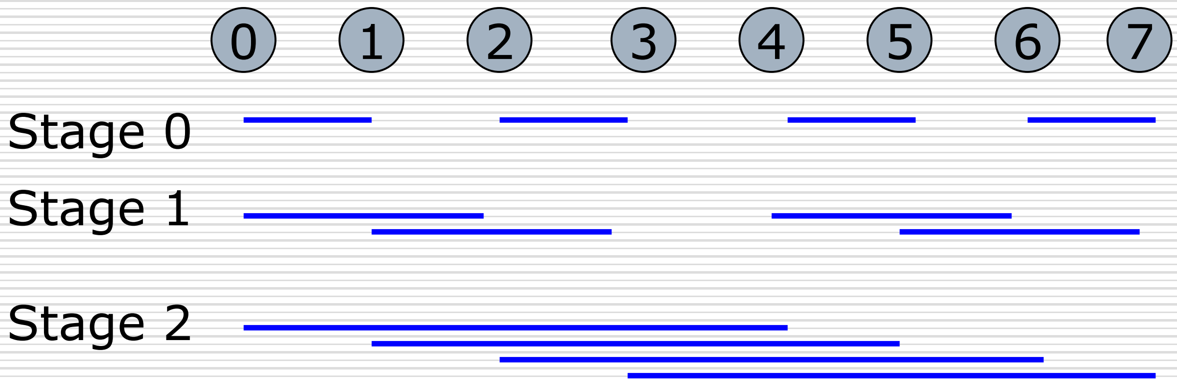
MPI_BARRIER(comm)



A return from barrier in one process tells the process that the other processes have ***entered*** the barrier.

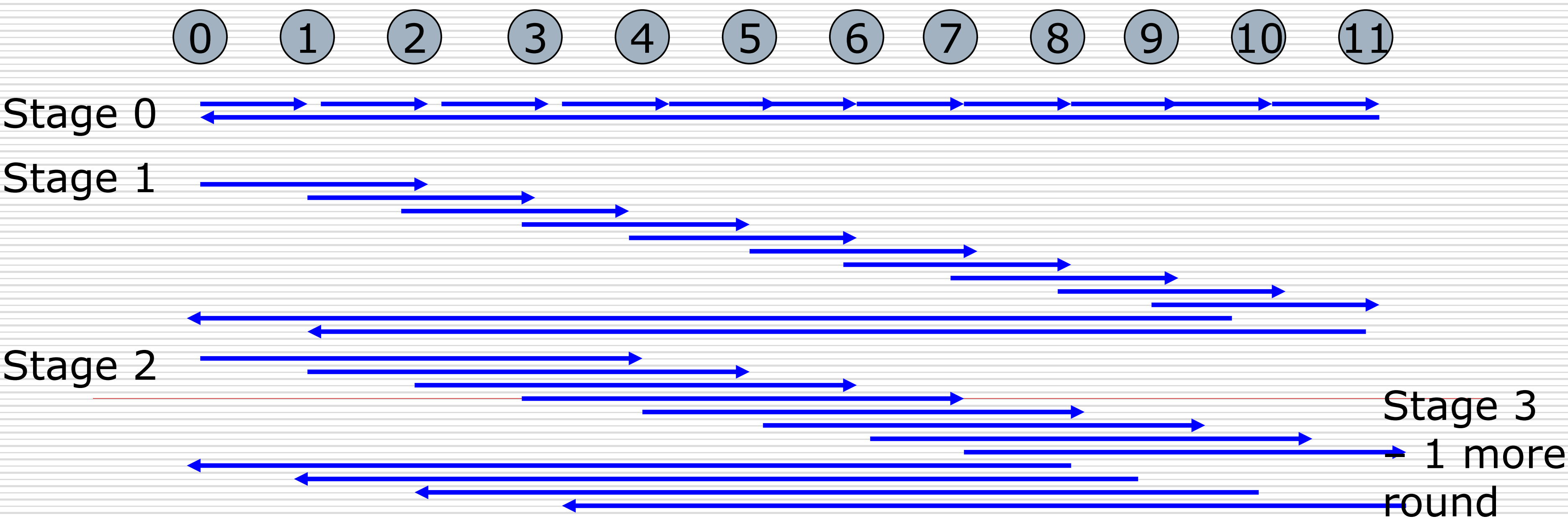
Barrier Implementation

- **Butterfly barrier** by Eugene Brooks II
- In round k, i synchronizes with $i \oplus 2^k$ pairwise.
- Worstcase – 2logP pairwise synchronizations by a processor



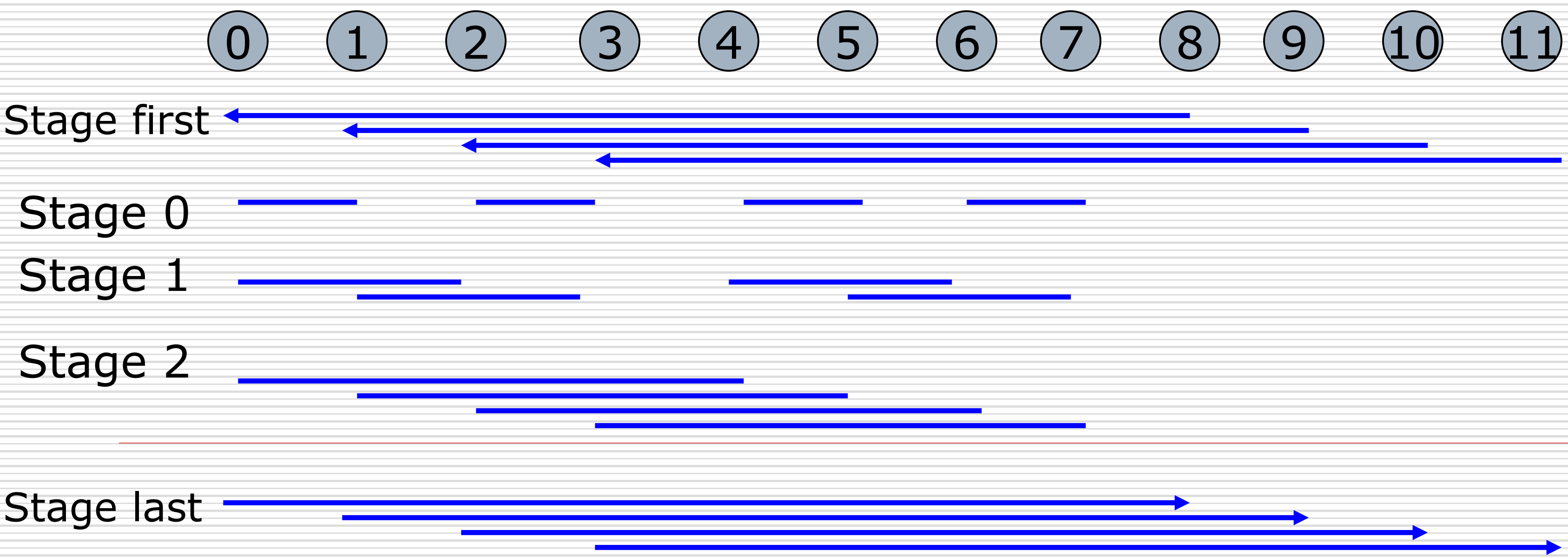
Barrier Algorithms

- **Dissemination barrier** by Hensgen, Finkel and Manser
- In round k, i signals $(i+2^k) \bmod P$
- No pairwise synchronization
- Atmost log(next power of 2 > P) on critical path irrespective of P



Barrier Algorithms

- **MPICH Barrier (pairwise exchange with recursive doubling)**
- Same as butterfly barrier.
- If nodes not equal to power, find the nearest power of 2, i.e. $m = 2^n$
- The last surfeit nodes, i.e. surfeit = size – m, initially send messages to the first surfeit number of nodes
- The first m nodes then perform butterfly barrier
- Finally, the first surfeit nodes send messages to the last surfeit nodes



AlltoAll

- The naive implementation

```
for all procs. i in order{
  if i # my proc., then send to i and recv from i
}
```

- MPICH implementation – similar to naïve, but doesn't do it in order

```
for all procs. i in order{
  dest = (my_proc+i)modP
  src = (myproc-i+P)modP
  send to dest and recv from src
}
```

PARALLEL SORTING

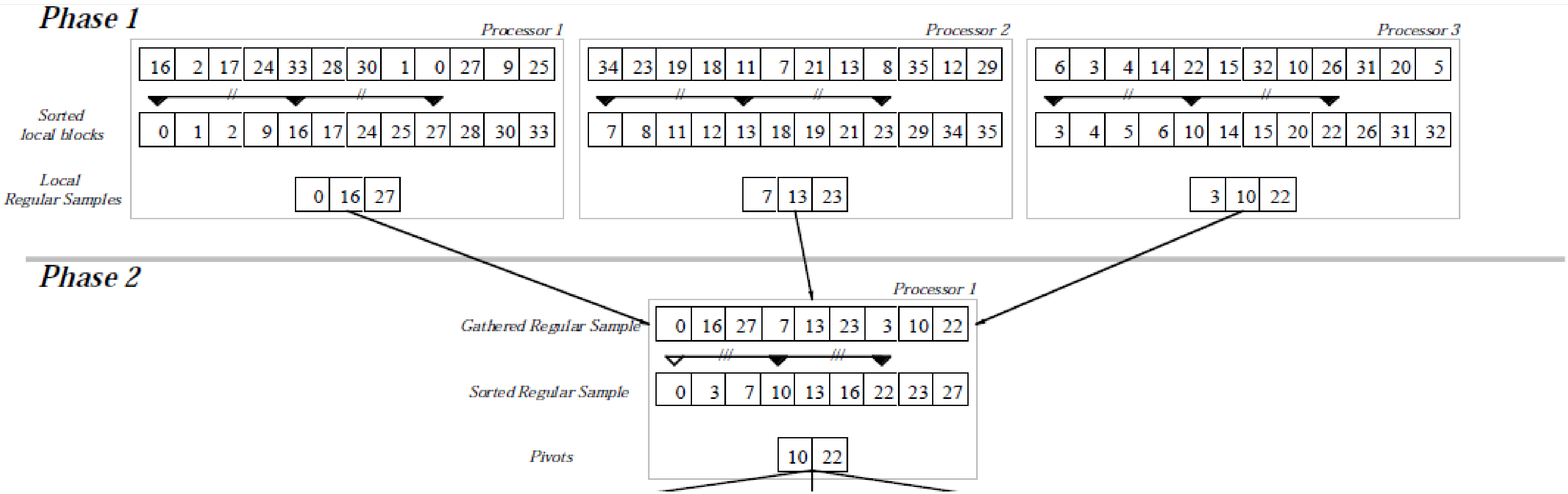
Introduction

- The input sequence of size N is distributed across P processors
- The output is such that elements in P_i is greater than elements in P_{i-1} and lesser than elements in P_{i+1}

Parallel Sorting by Regular Sampling (PSRS)

1. Each processor sorts its local data
2. Each processor selects a sample vector of size p-1; kth element is (n/p * (k+1)/p)
3. Samples are sent and merge-sorted on processor 0
4. Processor 0 defines a vector of p-1 *splitters* starting from p/2 element; i.e., kth element is p(k+1/2); broadcasts to the other processors

Example



Phase 1
Processor 1
16 2 17 24 33 28 30 1 0 27 9 25
Sorted local blocks
0 1 2 9 16 17 24 25 27 28 30 33
Local Regular Samples
0 16 27
Processor 2
34 23 19 18 11 7 21 13 8 35 12 29
7 8 11 12 13 18 19 21 23 29 34 35
7 13 23
Processor 3
6 3 4 14 22 15 32 10 26 31 20 5
3 4 5 6 10 14 15 20 22 26 31 32
3 10 22
Phase 2
Processor 1
Gathered Regular Sample
0 16 27 7 13 23 3 10 22
Sorted Regular Sample
0 3 7 10 13 16 22 23 27
Pivots
10 22

PSRS

5. Each processor sends local data to correct destination processors based on splitters; all-to-all exchange
6. Each processor merges the data chunk it receives

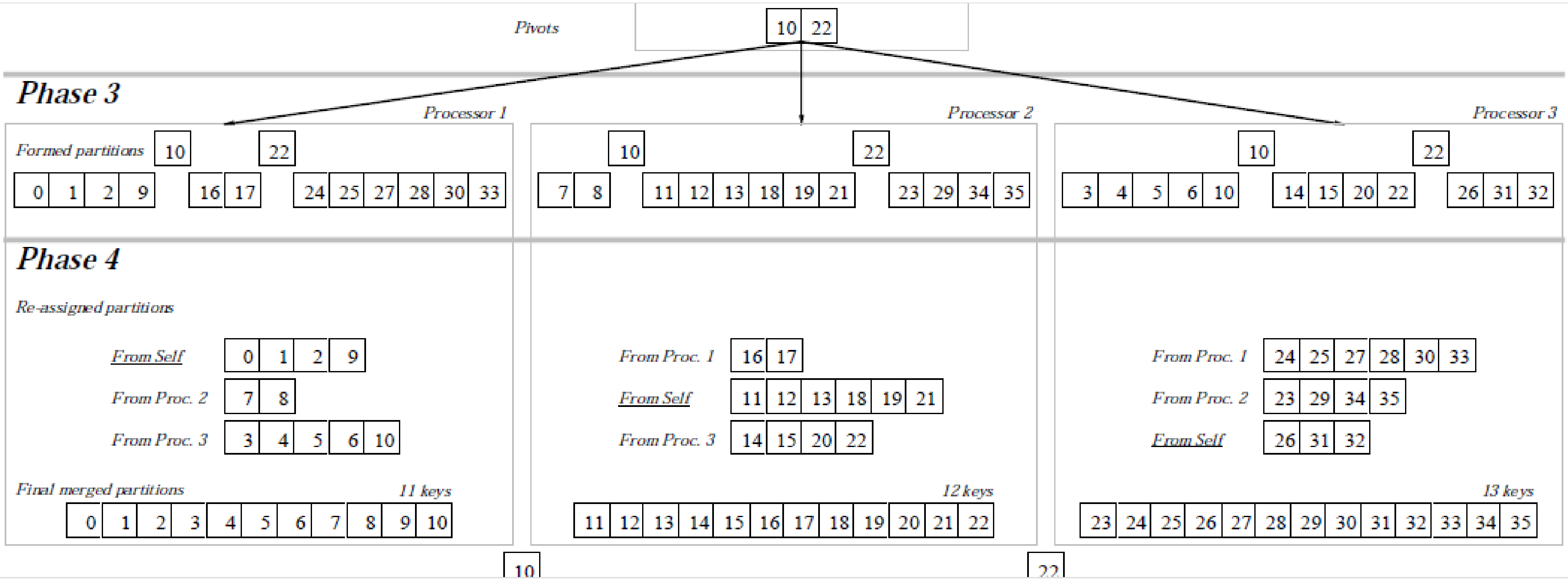
Step 5

- Each processor finds where each of the p-1 pivots divides its list, using a binary search
- i.e., finds the index of the largest element number larger than the jth pivot
- At this point, each processor has p sorted sublists with the property that each element in sublist i is greater than each element in sublist i-1 in any processor

Step 6

- Each processor i performs a p-way merge-sort to merge the ith sublists of p processors

Example Continued



Analysis

- The first phase of local sorting takes $O((n/p)\log(n/p))$
- 2nd phase:
  - Sorting p(p-1) elements in processor 0 – $O(p^2 \log p^2)$
  - Each processor performs p-1 binary searches of n/p elements – $p\log(n/p)$
- 3rd phase: Each processor merges (p-1) sublists
  - Size of data merged by any processor is no more than 2n/p (proof)
  - Complexity of this merge sort $2(n/p)\log p$
- Summing up: $O((n/p)\log n)$

Analysis

- 1st phase – no communication
- 2nd phase – p(p-1) data collected; p-1 data broadcast
- 3rd phase: Each processor sends (p-1) sublists to other p-1 processors; processors work on the sublists independently

- Graph Algorithms

Graph Traversal

- Graph search plays an important role in analyzing large data sets
- Relationship between data objects represented in the form of graphs
- Breadth first search used in finding shortest path or sets of paths

Parallel BFS
Level-synchronized algorithm

- Proceeds level-by-level starting with the source vertex
- Level of a vertex – its graph distance from the source
- Also, called **frontier-based** algorithm
- The parallel processes process a level, synchronize at the end of the level, before moving to the next level – Bulk Synchronous Parallelism (**BSP**) model
- How to decompose the graph (vertices, edges and adjacency matrix) among processors?

Distributed BFS with 1D Partitioning

- Each vertex and edges emanating from it are owned by one processor
- 1-D partitioning of the adjacency matrix

$$\begin{bmatrix} A_1 \\ \hline A_2 \\ \hline \vdots \\ \hline A_P \end{bmatrix}$$

- Edges emanating from vertex v is its edge list = list of vertex indices in row v of adjacency matrix *A*

1-D Partitioning

- At each level, each processor owns a set F – set of frontier vertices owned by the processor
- Edge lists of vertices in F are merged to form a set of neighboring vertices, N
- Some vertices of N owned by the same processor, while others owned by other processors
- Messages are sent to those processors to add these vertices to their frontier set for the next level

Algorithm 1 Distributed Breadth-First Expansion with 1D Partitioning

Initialize $L_{v_s}(v) = \begin{cases} 0, & v = v_s \text{, where } v_s \text{ is a source} \\ \infty, & \text{otherwise} \end{cases}$

for $l = 0$ to ∞ **do**

  $F \leftarrow \{v \mid L_{v_s}(v) = l\}$, the set of local vertices with level l

  if $F = \emptyset$ for all processors **then**

    Terminate main loop

  end if

  $N \leftarrow$ {neighbors of vertices in F (not necessarily local)}

  for all processors q **do**

    $N_q \leftarrow$ {vertices in N owned by processor q}

    Send N_q to processor q

    Receive $\bar{N}_q$ from processor q

  end for

  $\bar{N} \leftarrow \bigcup_q \bar{N}_q$ (The $\bar{N}_q$ may overlap)

  for $v \in \bar{N}$ and $L_{v_s}(v) = \infty$ **do**

    $L_{v_s}(v) \leftarrow l + 1$

  end for

end for

$L_{vs}(v)$ – level of v, i.e, graph distance from source vs

Parallel Depth First Search

- Easy to parallelize
- Left subtree can be searched in parallel with the right subtree
- Statically assign a node to a processor – the whole subtree rooted at that node can be searched independently.

Maintaining Search Space

- Each processor searches the space depth-first
- Unexplored states saved as stack; each processor maintains its own local stack
- Initially, the entire search space assigned to one processor
- The stack is then divided and distributed to processors

Termination Detection

- As processors search independently, how will they know when to terminate the program?
- Dijikstra's Token Termination Detection Algorithm
  - Based on passing of a token in a logical ring; P0 initiates a token when idle; A processor holds a token until it has completed its work, and then passes to the next processor; when P0 receives again, then all processors have completed

Tree Based Termination Detection

- Uses weights
- Initially processor 0 has weight 1
- When a processor transfers work to another processor, the weights are halved in both the processors
- When a processor finishes, weights are returned
- Termination is when processor 0 gets back 1
- Goes with the DFS algorithm; No separate communication steps

- Combinatorial algorithms - APSP

All-Pairs Shortest Paths Floyd's Algorithm

- Consider a subset S = {v1,v2,…,vk} of vertices for some k <= n
- Consider finding shortest path between vi and vj
- Consider all paths from vi to vj whose intermediate vertices belong to the set S; Let $p_{i,j}^{(k)}$ be the minimum-weight path among them with weight $d_{i,j}^{(k)}$

All-Pairs Shortest Paths Floyd's Algorithm

- If vk is not in the shortest path, then $p_{i,j}^{(k)} = p_{i,j}^{(k-1)}$
- If vk is in the shortest path, then the path is broken into two parts – from vi to vk, and from vk to vj
- So $d_{i,j}^{(k)} = \min\{d_{i,j}^{(k-1)}, d_{i,k}^{(k-1)} + d_{k,j}^{(k-1)}\}$
- The length of the shortest path from vi to vj is given by $d_{i,j}^{(n)}$.
- In general, solution is a matrix $D^{(n)}$

Parallel Formulation
2-D Block Mapping

- Processors laid in a 2D mesh
- During kth iteration, each process Pi,j needs certain segments of the kth row and kth column of the D(k-1) matrix
- For $d_{l,r}^{(k)}$: following are needed
  - $d_{l,k}^{(k-1)}$ (from a process along the same process row)
  - $d_{k,r}^{(k-1)}$ (from a process along the same process column)

Parallel Formulation
2D Block Mapping

- During kth iteration, each of the root(p) processes containing part of the kth row sends it to root(p)-1 in same column;
- Similarly for the same row