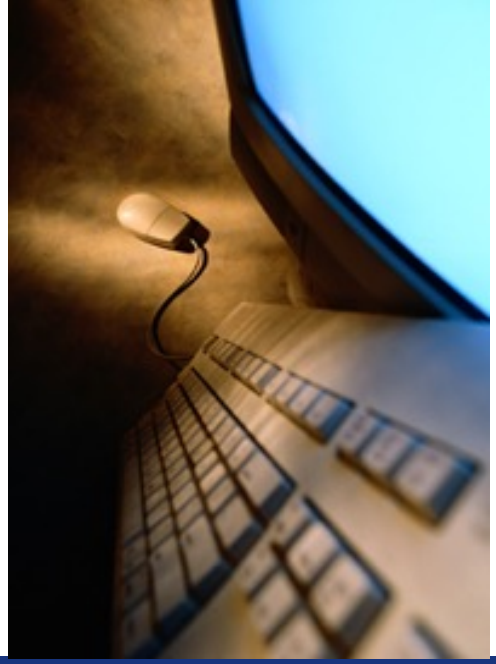
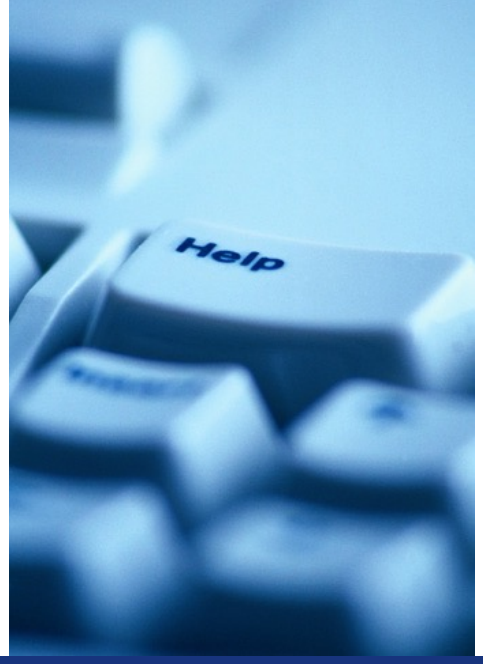
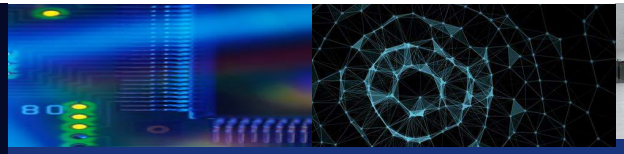



并行与分布式计算

Parallel & Distributed Computing

陈鹏飞
计算机学院
chenpf7@mail.sysu.edu.cn



Lecture 14 — Parallel Graph Algorithms

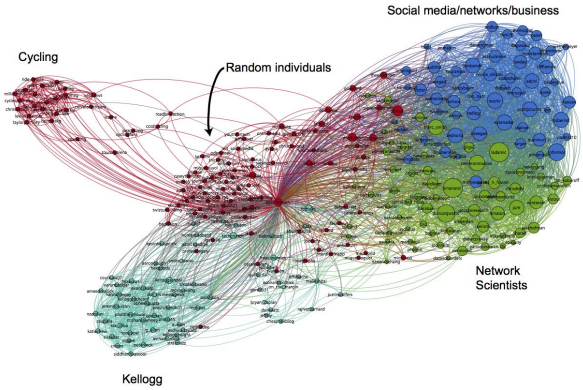
Pengfei Chen

School of Computer Science and Engineering

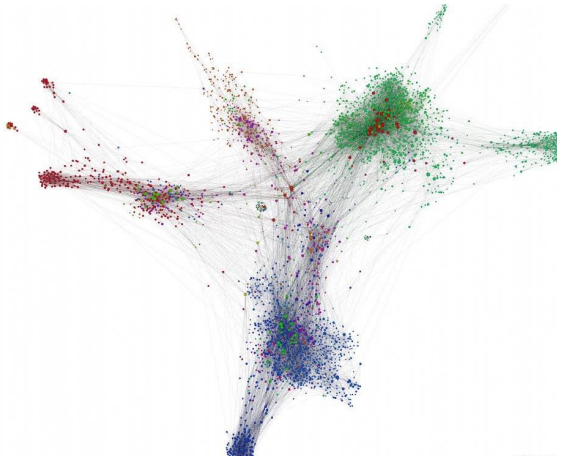
chenpf7@mail.sysu.edu.cn

Reference to Chapter 10 of "Introduction to Parallel Computing," Addison Wesley, the 2nd version, 2003

Scenario

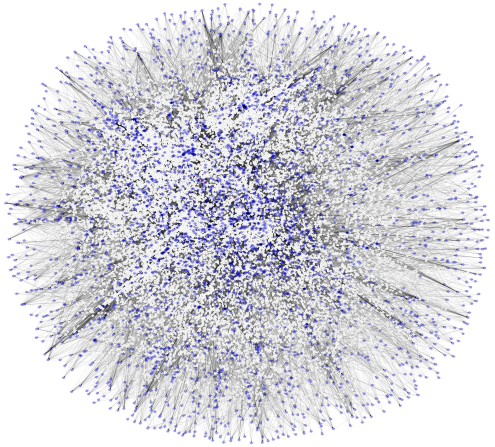


Social Graph



Proteins Regular Graph

Service dependency Graph

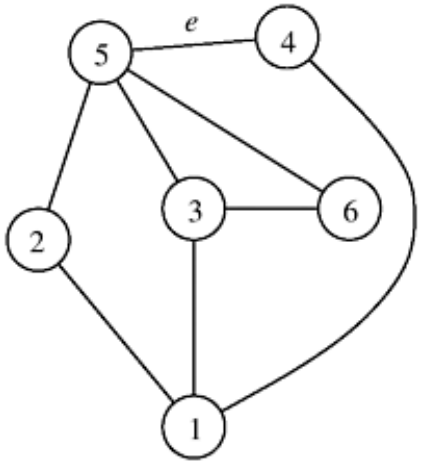


Alert Impact Graph

Graph definition

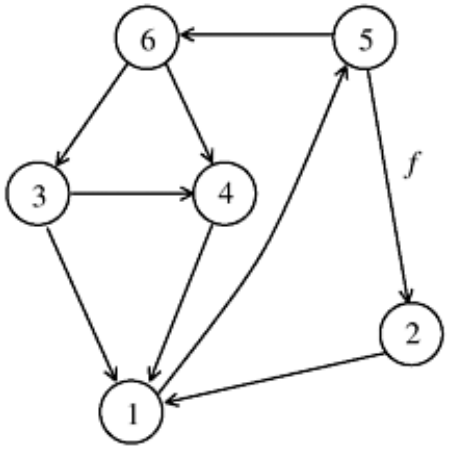
- An ***undirected graph*** G is a pair (V, E), where V is a finite set of points called ***vertices*** and E is a finite set of ***edges***. An edge $e \in E$ is an unordered pair (u, v), where $u, v \in V$. An edge (u, v) indicates that vertices u and v are connected.

- Similarly, a ***directed graph*** G, is a pair (V, E), where V is the set of vertices as we just defined, but an edge $(u, v) \in E$ is an ordered pair; that is, it indicates that there is a connection from u to v.



(a)

undirected



(b)

directed

In this lecture

- **Parallel formulations of some important and fundamental graph algorithms**
  - **Graph theory plays an important role in CS**
  - **It provides an easy and systematic way to model many problems**
  - **Many problems can be expressed in terms of graphs, and can be solved using standard graph algorithms**

Topics

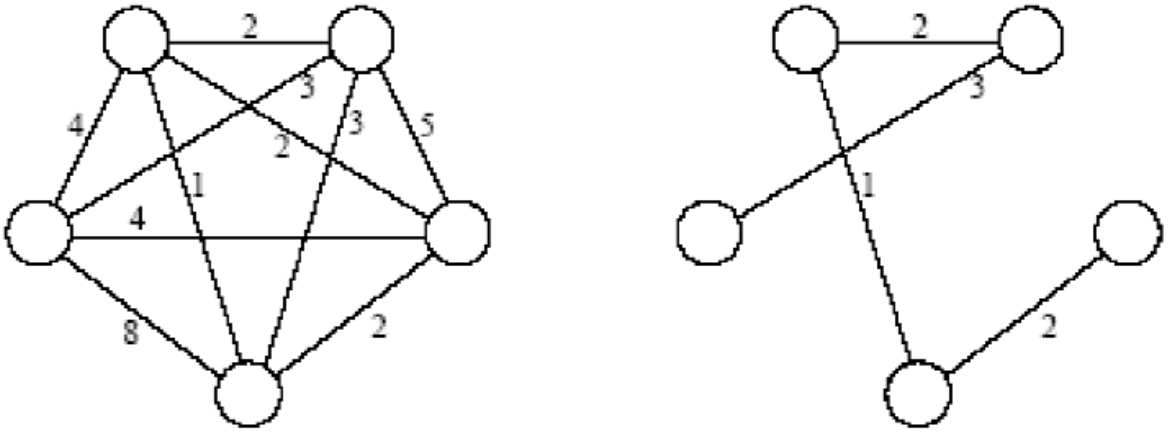
- **Minimum Spanning Tree: Prim's Algorithm**
- **Single-Source Shortest Paths: Dijkstra's Algorithm**
- **All-Pairs Shortest Paths**
- **Connected Components**

Minimum Spanning Tree

- A spanning tree of an undirected graph G is a subgraph of G that is a tree containing all the vertices of G.
- In a weighted graph, the weight of a subgraph is the sum of the weights of the edges in the subgraph.
- A minimum spanning tree (MST) for a weighted undirected graph is a spanning tree with minimum weight.

A. Grama et al., “Introduction to Parallel Computing,” Addison Wesley, 2003

Minimum Spanning Tree



An undirected graph and its minimum spanning tree.

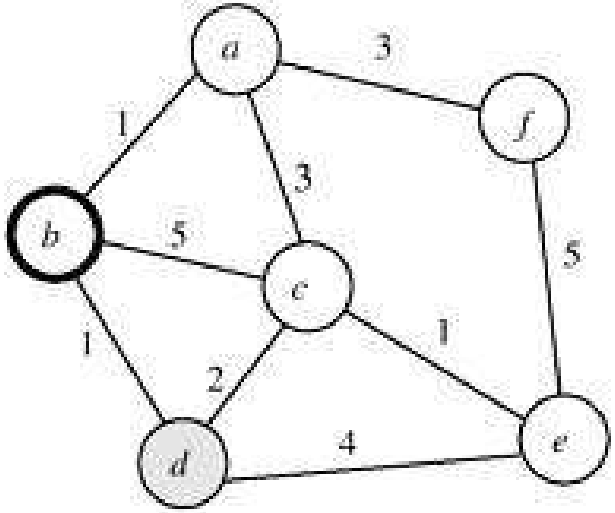
A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

Minimum Spanning Tree: Prim's Algorithm

- **Prim's algorithm for finding an MST is a greedy algorithm.**
- **Start by selecting an arbitrary vertex, include it into the current MST.**
- **Grow the current MST by inserting into it the vertex closest to one of the vertices already in current MST.**

Minimum Spanning Tree: Prim's Algorithm

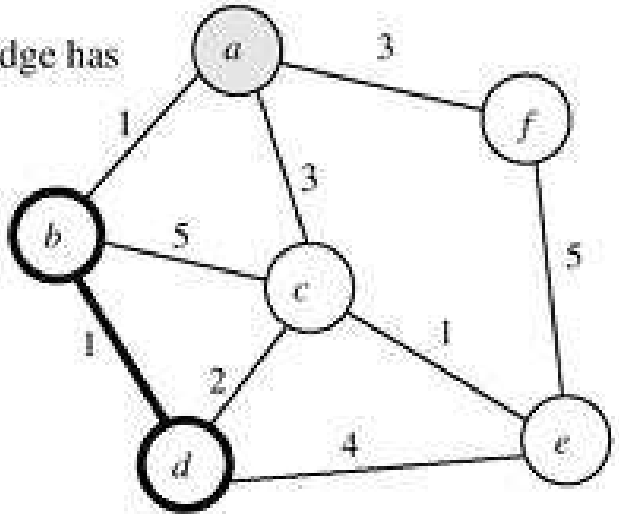
(a) Original graph



	a	b	c	d	e	f
d[ ]	1	0	5	1	∞	∞

	a	b	c	d	e	f
a	0	1	3	∞	∞	3
b	1	0	5	1	∞	∞
c	3	5	0	2	1	∞
d	∞	1	2	0	4	∞
e	∞	∞	1	4	0	5
f	2	∞	∞	∞	5	0

(b) After the first edge has been selected

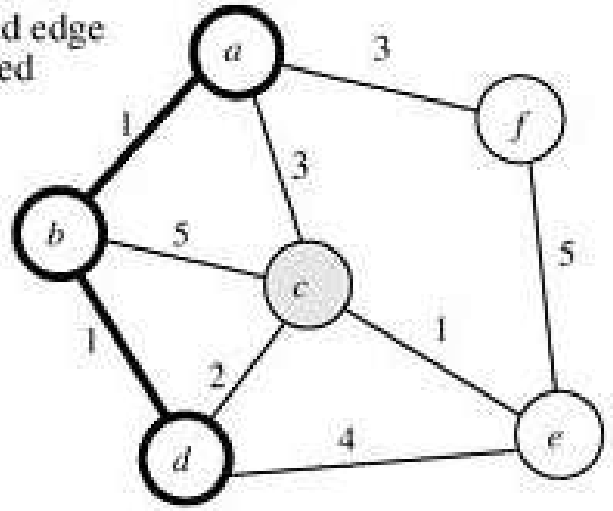


	a	b	c	d	e	f
d[ ]	1	0	2	1	4	∞

	a	b	c	d	e	f
a	0	1	3	∞	∞	3
b	1	0	5	1	∞	∞
c	3	5	0	2	1	∞
d	∞	1	2	0	4	∞
e	∞	∞	1	4	0	5
f	2	∞	∞	∞	5	0

Minimum Spanning Tree: Prim's Algorithm

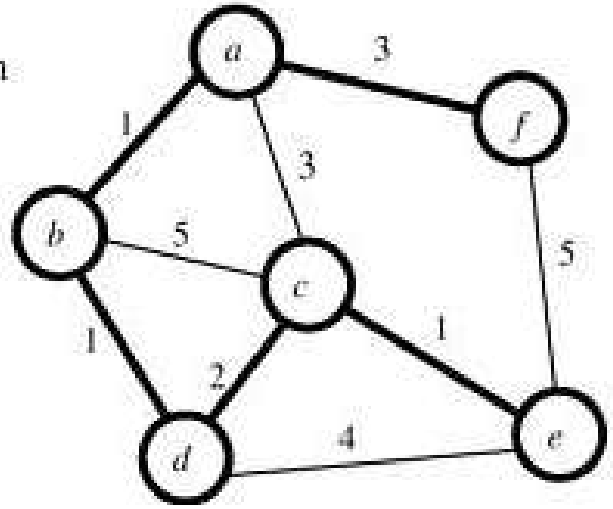
(c) After the second edge has been selected



	a	b	c	d	e	f
d[ ]	1	0	2	1	4	3

	a	b	c	d	e	f
a	0	1	3	∞	∞	3
b	1	0	5	1	∞	∞
c	3	5	0	2	1	∞
d	∞	1	2	0	4	∞
e	∞	∞	1	4	0	5
f	2	∞	∞	∞	5	0

(d) Final minimum spanning tree



	a	b	c	d	e	f
d[ ]	1	0	2	1	1	3

	a	b	c	d	e	f
a	0	1	3	∞	∞	3
b	1	0	5	1	∞	∞
c	3	5	0	2	1	∞
d	∞	1	2	0	4	∞
e	∞	∞	1	4	0	5
f	2	∞	∞	∞	5	0

Minimum Spanning Tree: Prim's Algorithm

```
1.    procedure PRIM_MST(V, E, w, r)
2.    begin
3.        V_T := {r};
4.        d[r] := 0;
5.        for all v ∈ (V − V_T) do
6.            if edge (r, v) exists set d[v] := w(r, v);
7.            else set d[v] := ∞;
8.        while V_T ≠ V do
9.        begin
10.           find a vertex u such that d[u] := min{d[v]|v ∈ (V − V_T)};
11.           V_T := V_T ∪ {u};
12.           for all v ∈ (V − V_T) do
13.               d[v] := min{d[v], w(u, v)};
14.       endwhile
15.   end PRIM_MST
```

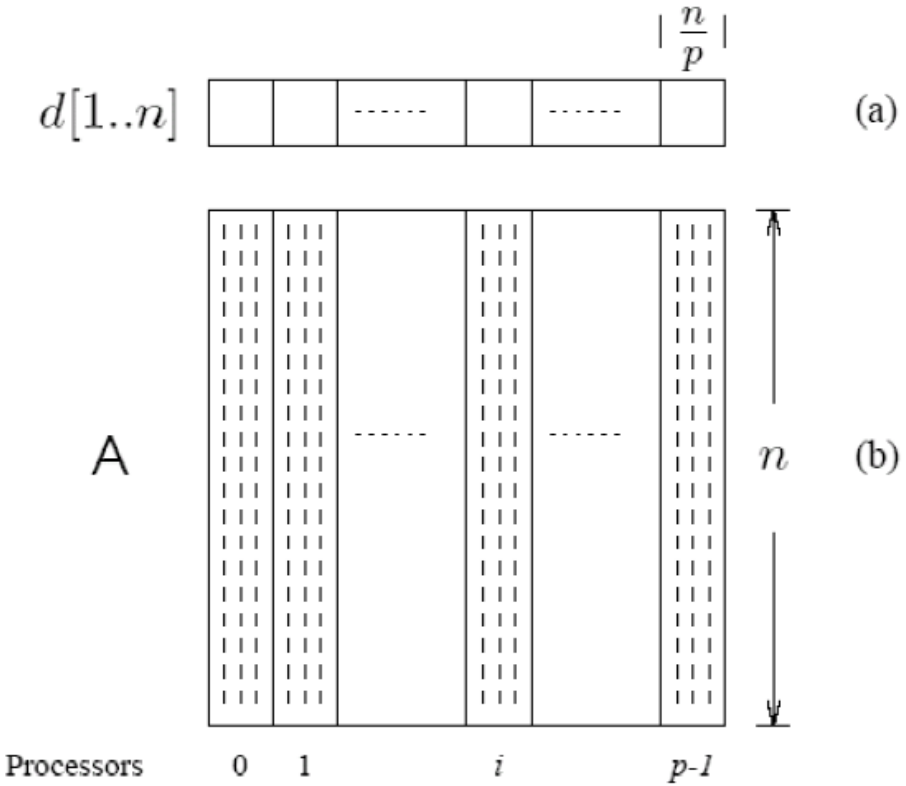
Prim's sequential minimum spanning tree algorithm

A. Grama et al., “Introduction to Parallel Computing,” Addison Wesley, 2003

Prim's Algorithm: Parallel Formulation

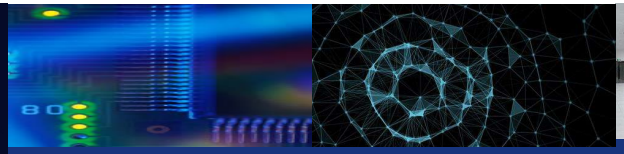
- **The algorithm works in n outer iterations - it is hard to execute these iterations concurrently.**
- **The inner loop is relatively easy to parallelize**
  - **Let p be the number of processes, and let n be the number of vertices.**
- **The adjacency matrix is partitioned in a 1-D block fashion, with distance vector d partitioned accordingly.**
- **In each step, a processor selects the locally closest node, followed by a global reduction to select globally closest node.**
- **This node is inserted into MST, and the choice broadcast to all processors. Each processor updates its part of the d vector locally.**

Prim's Algorithm: Parallel Formulation



The partitioning of the distance array *d* and the adjacency matrix *A* among *p* processes.

A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003



Prim's Algorithm: Parallel Formulation

- **The parallel time per iteration is $O(n/p + \log p)$.**
  - **The cost of a broadcast is $O(\log p)$.**
  - **The cost of local update of the d vector is $O(n/p)$.**
- **The total parallel time is given by $O(n^2/p + n \log p)$.**

- **ATTENSION**
  - **$O(n/p)$ for minimum selection and local update is only estimated for the algorithm we use here, which can be improved by a better algorithm and advance data structures (e.g., heap)**

Topics

- Minimum Spanning Tree: Prim's Algorithm
- Single-Source Shortest Paths: Dijkstra's Algorithm
- All-Pairs Shortest Paths
- Connected Components
- Algorithms for Sparse Graphs

Single-Source Shortest Paths

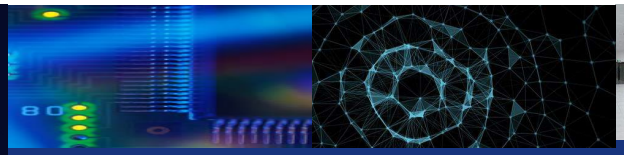
- **For a weighted graph $G = (V,E,w)$, the *single-source all-sinks shortest paths* problem is to find the shortest paths from a vertex $v \in V$ to all other vertices in V.**
- **Dijkstra's algorithm is similar to Prim's algorithm. It maintains a set of nodes for which the shortest paths are known.**
- **It grows this set based on the node closest to source using one of the nodes in the current shortest path set.**

Single-Source Shortest Paths: Dijkstra's Algorithm

```
1.    procedure DIJKSTRA_SINGLE_SOURCE_SP(V, E, w, s)
2.    begin
3.        V_T := {s};
4.        for all v ∈ (V − V_T) do
5.            if (s, v) exists set l[v] := w(s, v);
6.            else set l[v] := ∞;
7.        while V_T ≠ V do
8.        begin
9.            find a vertex u such that l[u] := min{l[v]|v ∈ (V − V_T)};
10.           V_T := V_T ∪ {u};
11.           for all v ∈ (V − V_T) do
12.               l[v] := min{l[v], l[u] + w(u, v)};
13.       endwhile
14.   end DIJKSTRA_SINGLE_SOURCE_SP
```

Dijkstra's sequential single-source shortest paths algorithm.

A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

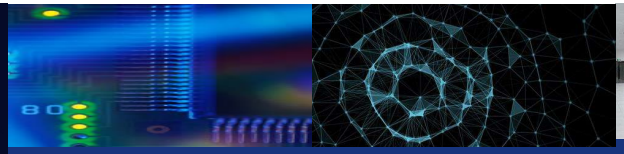


Dijkstra's Algorithm: Parallel Formulation

- **Very similar to the parallel formulation of Prim's algorithm for minimum spanning trees.**
  - **The weighted adjacency matrix is partitioned using the 1-D block mapping.**
  - **Each process selects, locally, the node closest to the source, followed by a global reduction to select next node.**
  - **The node is broadcast to all processors and the *l*-vector updated.**
- **The parallel performance of Dijkstra's algorithm is identical to that of Prim's algorithm.**

Topics

- Minimum Spanning Tree: Prim's Algorithm
- Single-Source Shortest Paths: Dijkstra's Algorithm
- All-Pairs Shortest Paths
- Connected Components



All-Pairs Shortest Paths

- **Given a weighted graph $G(V,E,w)$, the *all-pairs shortest paths* problem is to find the shortest paths between all pairs of vertices $v_i, v_j \in V$.**
- **A number of algorithms are known for solving this problem.**

A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

Dijkstra's Algorithm

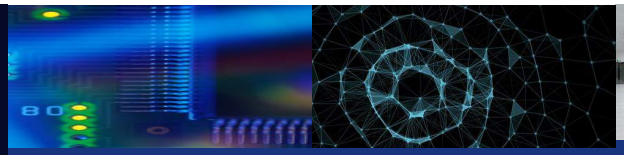
- **Execute *n* instances of the single-source shortest path problem, one for each of the *n* source vertices.**
- **Complexity is $O(n^3)$.**

Dijkstra's Algorithm

- **Two parallelization strategies**
  - **(source partitioned) execute each of the *n* shortest path problems on a different processor, or**
  - **(source parallel) use a parallel formulation of the shortest path problem to increase concurrency**

Dijkstra's Algorithm (Source Partitioned)

- **Source-by-source parallelism**
- **Use p processes to execute the Dijkstra's SSSP algorithm on n vertices ($p \leq n$)**
  - **no interprocess communication needed, assuming the adjacency matrix is replicated at all processes**
  - **The parallel runtime is $\Theta(n^3/p)$**
- **The maximum #proc is $\Theta(n)$**
  - **with runtime is $\Theta(n^2)$**



Dijkstra's Algorithm (Source Parallel)

- **What if $p > n$ …**
  - **each of the SSSP is further executed in parallel**
  - **parallel execution of SSSP on n vertices**
  - **use p/n processes to execute SSSP for each vertex**
- **Using previous results, this takes total time:**

$$T_P = \overbrace{\Theta\left(\frac{n^3}{p}\right)}^{\text{computation}} + \overbrace{\Theta(n \log p)}^{\text{communication}}.$$

  - **($\log p$ is a pessimistic estimate of the broadcasting time)**
- **The maximum #proc is $\Theta(n^2/\log n)$**
  - **with runtime $\Theta(n \log n)$**
  - **when the efficiency is not degraded**

$$S = \frac{\Theta(n^3)}{\Theta(n^3/p) + \Theta(n \log p)}$$

$$E = \frac{1}{1 + \Theta((p \log p)/n^2)}$$

A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

Floyd's Algorithm

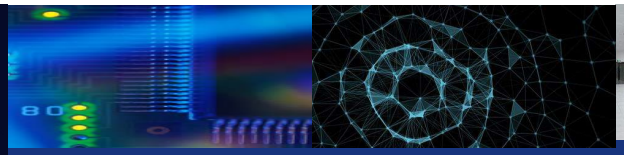
The following recurrence relation follows:

$$d_{i,j}^{(k)} = \begin{cases} w(v_i, v_j) & \text{if } k = 0 \\ \min\left\{ d_{i,j}^{(k-1)}, d_{i,k}^{(k-1)} + d_{k,j}^{(k-1)} \right\} & \text{if } k \geq 1 \end{cases}$$

Computed for each pair of nodes and for $k = 1, \ldots, n$.

The serial complexity is $O(n^3)$.

A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003



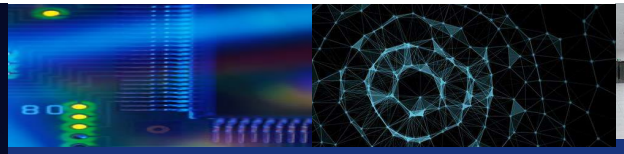
Floyd's Algorithm

```
1.    procedure FLOYD_ALL_PAIRS_SP(A)
2.    begin
3.        D^(0) = A;
4.        for k := 1 to n do
5.            for i := 1 to n do
6.                for j := 1 to n do
7.                    d_{i,j}^(k) := min(d_{i,j}^(k-1), d_{i,k}^(k-1) + d_{k,j}^(k-1));
8.    end FLOYD_ALL_PAIRS_SP
```

Floyd's all-pairs shortest paths algorithm. This program computes the all-pairs shortest paths of the graph $G = (V,E)$ with adjacency matrix A.

Floyd's Algorithm (1-D Block Mapping)

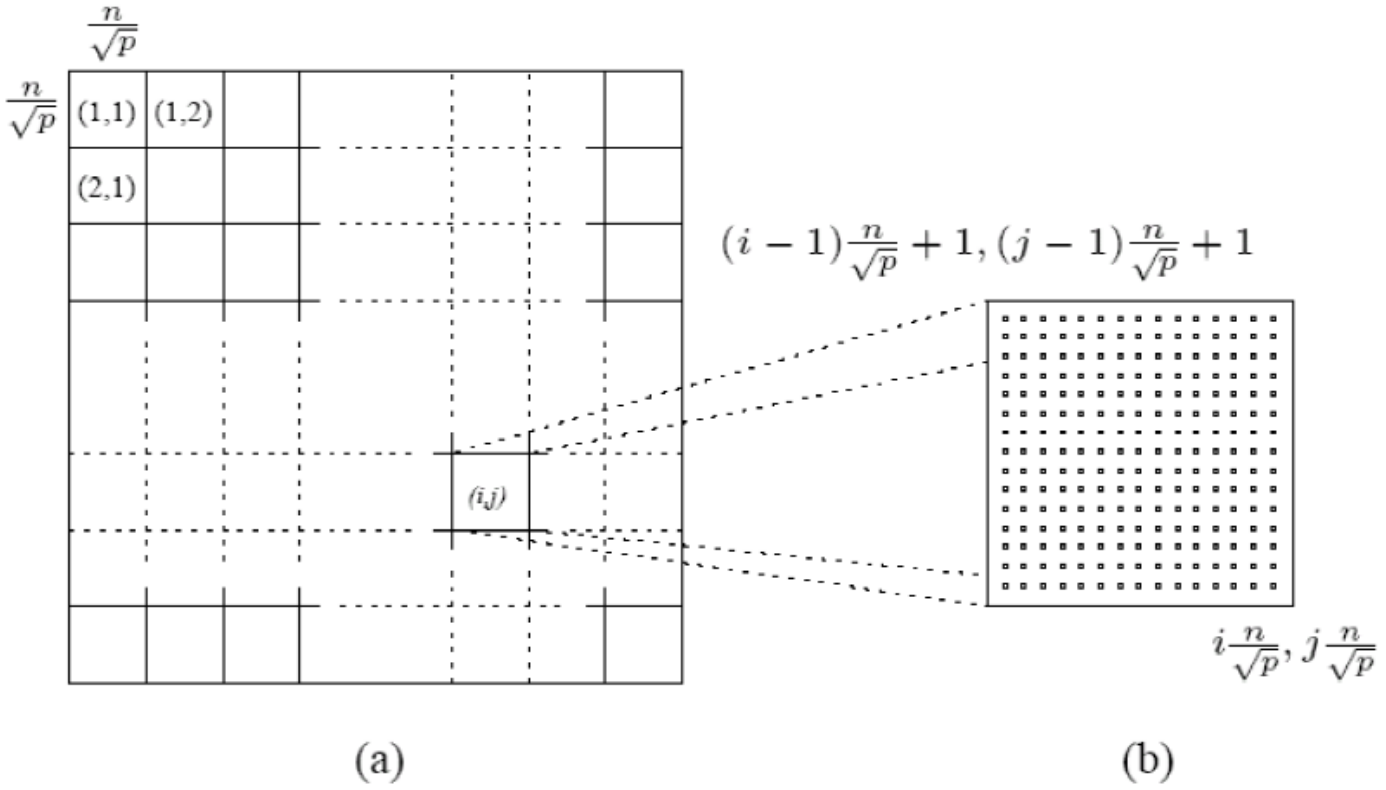
- **Matrix $D^{(k)}$ is divided into p blocks with size $(n/p) \times n$**
  - **Each process update its part of the matrix ($p \leq n$)**
- **To compute** $d_{i,j}^{(k)} = \min\left(d_{i,j}^{(k-1)}, d_{i,k}^{(k-1)} + d_{k,j}^{(k-1)}\right)$
  - $d_{i,j}^{(k-1)}$ **and** $d_{i,k}^{(k-1)}$ **is owned by** P_i **itself**
  - $d_{k,j}^{(k-1)}$ **is owned by** P_k
- **In general, during the k^{th} iteration, the processes containing the k^{th} row is responsible for broadcast**
- **Parallel runtime is $\Theta(n^3/p + n^2 \log p)$**
- **The maximum #proc is $\Theta(n/\log n)$**
  - **with runtime $\Theta(n^2 \log n)$**
  - **when the efficiency is not degraded**



Floyd's Algorithm (2-D Block Mapping)

- **Matrix $D^{(k)}$ is divided into p blocks of size $(n/\sqrt{p}) \times (n/\sqrt{p})$.**
  - **Each processor updates its part of the matrix during each iteration.**
  - **To compute $d_{i,j}^{(k)}$ processor $P_{i,j}$ must get $d_{i,k}^{(k-1)}$ and $d_{k,j}^{(k-1)}$.**
- **In general, during the k^{th} iteration**
  - **each of the $\sqrt{p}$ processes containing part of the k^{th} row send it to the $\sqrt{p}-1$ processes in the same column.**
  - **each of the $\sqrt{p}$ processes containing part of the k^{th} column sends it to the $\sqrt{p}-1$ processes in the same row.**

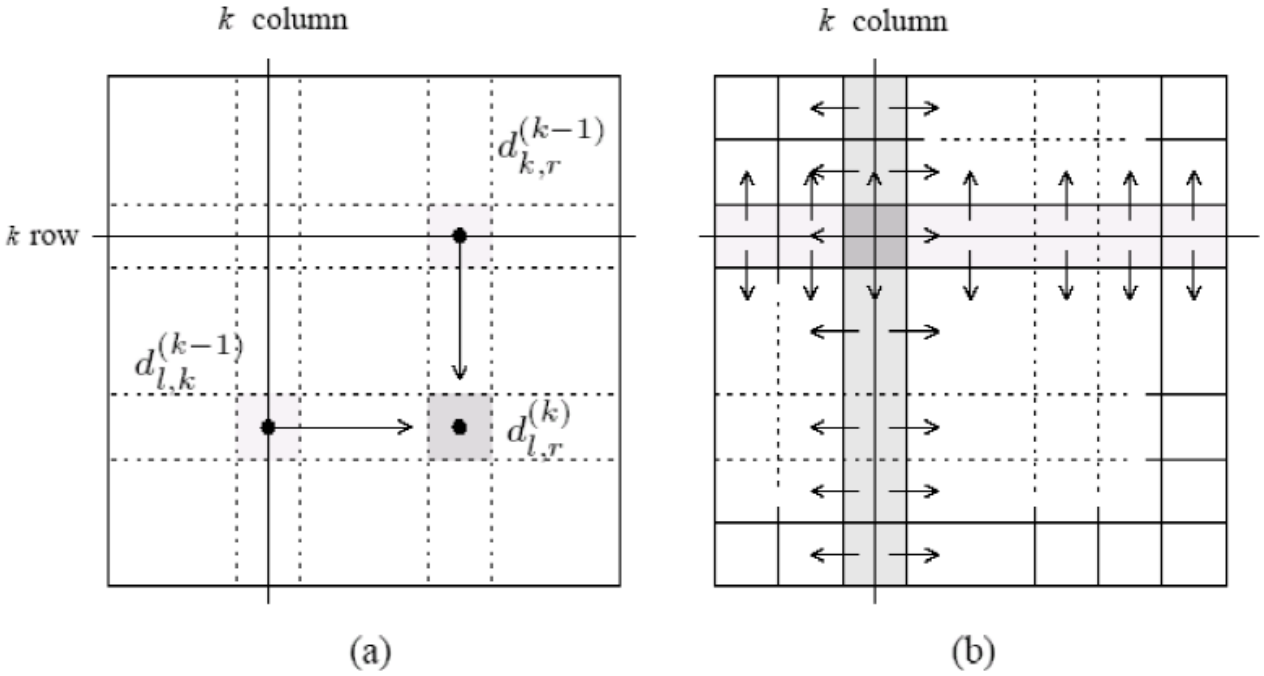
Floyd's Algorithm (2-D Block Mapping)



(a) Matrix $D^{(k)}$ distributed by 2-D block mapping into $\sqrt{p} \times \sqrt{p}$ subblocks, and (b) the subblock of $D^{(k)}$ assigned to process $P_{i,j}$.

A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

Floyd's Algorithm: Communications



(a) Communication patterns used in the 2-D block mapping. When computing $d_{i,j}^{(k)}$, information must be sent to the highlighted process from two other processes along the same row and column. (b) The row and column of $\sqrt{p}$ processes that contain the k^{th} row and column send them along process columns and rows.

A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

Floyd's Algorithm (2-D Block Mapping)

```
1.   procedure FLOYD_2DBLOCK(D^(0))
2.   begin
3.       for k := 1 to n do
4.       begin
5.           each process P_{i,j} that has a segment of the k^th row of D^(k-1);
                 broadcasts it to the P_{*,j} processes;
6.           each process P_{i,j} that has a segment of the k^th column of D^(k-1);
                 broadcasts it to the P_{i,*} processes;
7.           each process waits to receive the needed segments;
8.           each process P_{i,j} computes its part of the D^(k) matrix;
9.       end
10.  end FLOYD_2DBLOCK
```

Floyd's parallel formulation using the 2-D block mapping. $P_{*,j}$ denotes all the processes in the j^{th} column, and $P_{i,*}$ denotes all the processes in the i^{th} row. The matrix $D^{(0)}$ is the adjacency matrix.

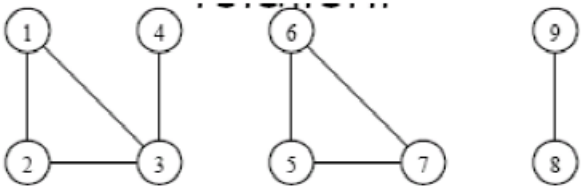
$$T_P = \overbrace{\Theta\left(\frac{n^3}{p}\right)}^{\text{computation}} + \overbrace{\Theta\left(\frac{n^2}{\sqrt{p}}\log p\right)}^{\text{communication}}.$$

Topics

- Minimum Spanning Tree: Prim's Algorithm
- Single-Source Shortest Paths: Dijkstra's Algorithm
- All-Pairs Shortest Paths
- Connected Components

Connected Components

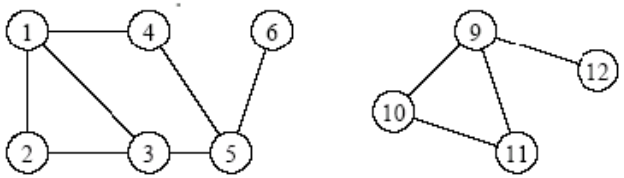
- **The connected components of an undirected graph are the equivalence classes of vertices under the "is reachable from" relation.**



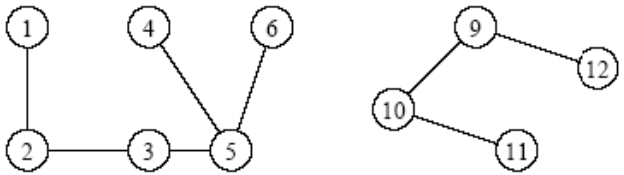
A graph with three connected components:
{1,2,3,4}, *{5,6,7}*, **and** *{8,9}*.

Connected Components: Depth-First Search Based Algorithm

- **Perform DFS on the graph to get a forest - each tree in the forest corresponds to a separate connected component.**



(a)



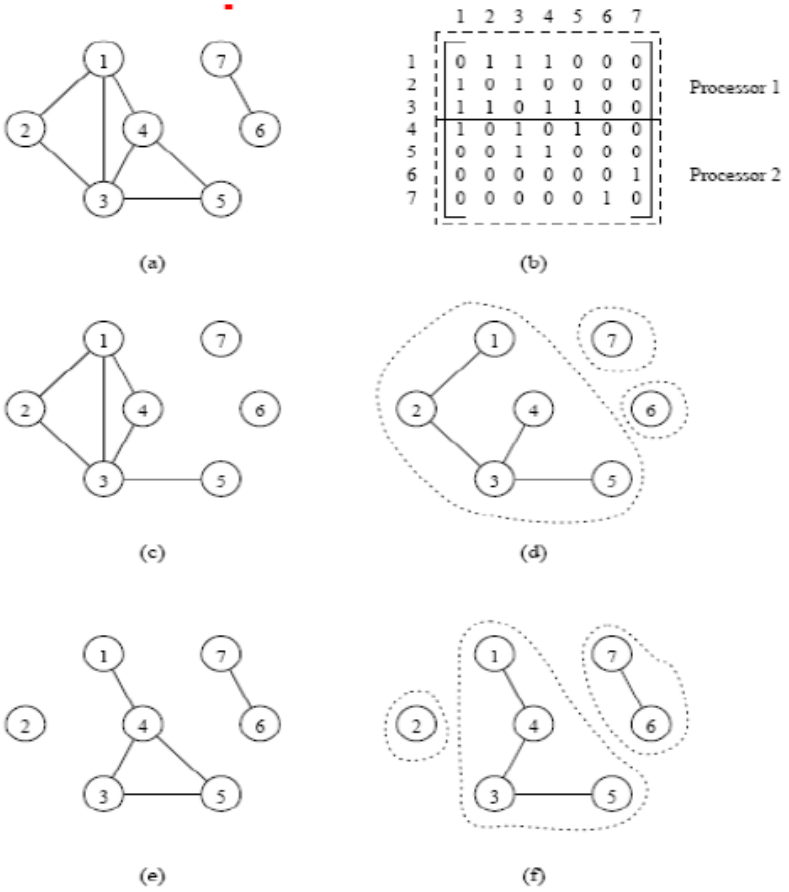
(b)

Part (b) is a depth-first forest obtained from depth-first traversal of the graph in part (a). Each of these trees is a connected component of the graph in part (a).

Connected Components: Parallel Formulation

- **Partition the graph across processes and run independent connected component algorithms on each processor. At this point, we have p spanning forests.**
- **In the second step, spanning forests are merged pairwise until only one spanning forest remains.**

Connected Components: Parallel Formulation



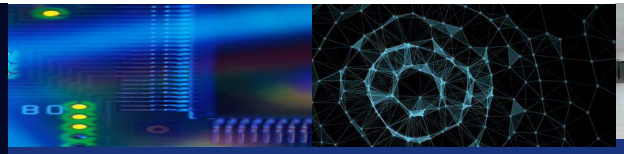
Computing connected components in parallel. The adjacency matrix of the graph *G* in (a) is partitioned into two parts (b). Each process gets a subgraph of *G* ((c) and (e)). Each process then computes the spanning forest of the subgraph ((d) and (f)). Finally, the two spanning trees are merged to form the solution.

Merge process

The spanning forests are merged as follows. Let *A* and *B* be the two spanning forests to be merged. At most *n* - 1 edges (since *A* and *B* are forests) of one are merged with the edges of the other. Suppose we want to merge forest *A* into forest *B*. For each edge (*u*, *v*) of *A*, a find operation is performed for each vertex to determine if the two vertices are already in the same tree of *B* . If not, then the two trees (sets) of *B* containing *u* and *v* are united by a union operation. Otherwise, no union operation is necessary. Hence, merging *A* and *B* requires at most 2(*n* - 1) find operations and (*n* - 1) union operations. We can implement the disjoint-set data structure by using disjoint-set forests with ranking and path compression. Using this implementation, the cost of performing 2(*n* - 1) finds and (*n* - 1) unions is *O* (*n*). A detailed description of the disjoint-set forest is beyond the scope of this book. Refer to the bibliographic remarks (Section 10.8) for references.

1-D Block Mapping The *n* x *n* adjacency matrix is partitioned into *p* stripes (Section 3.4.1). Each stripe is composed of *n*/*p* consecutive rows and is assigned to one of the *p* processes. To compute the connected components, each process first computes a spanning forest for the *n*-vertex graph represented by the *n*/*p* rows of the adjacency matrix assigned to it.

$$T_P = \overbrace{\Theta\left(\frac{n^2}{p}\right)}^{\text{local computation}} + \overbrace{\Theta(n \log p).}^{\text{forest merging}}$$



Thank You !