Parallel Distributed Shortest Paths

Ned Nedialkov

McMaster University
Canada

SE 3F03
March 2013

Outline

For more details see A. Grama, G. Karypis, V. Kumar, A. Gupta, Introduction to Parallel Computing

Shortest paths

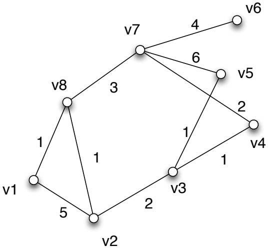
- $G = (V, E, w)$ weighted graph
- G can be directed or undirected
- V is a set of vertices
- E is a set of edges
- $w : E \rightarrow R^+$ is a weight function
- Given $s \in V$, find the shortest paths from s to all vertices in V
- Dijkstra's algoritm
  - Finds all shortest distances from s
  - Greedy algorithm: always choses the closest vertex
  - Original algorithm $O(|V|^2)$
  - Using a min-priority queue implemented by a Fibonacci heap $O(|E| + |V| \log |V|)$, Fredman & Tarjan 1984

Dijkstra's algoritm

Algorithm (Dijkstra)

Input: $G = (V, E, w)$, source s
Output: d an array of shortest distances
Compute:
% initialization
$V_T = \{s\}, d[s] = 0$
for all $v \in (V - V_T)$ **do**
  if $(s, v) \in E$ then $d[v] = w(s, v)$
  else $d[v] = \infty$
% find distances
while $V_T \neq V$ **do**
  find u such that $d[u] = \min\{d[v] \mid v \in V - V_T\}$
  $V_T = V_T \cup \{u\}$
  for all $v \in V - V_T$ do
    $d[v] = \min\{d[v], d[u] + w(u, v)\}$

Example



$s = v_1,\ V_T = \{v_1\}$
After the for loop
$d = (0, 5, \infty, \infty, \infty, \infty, \infty, 1)$

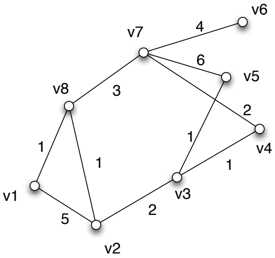
Iterations of the while loop

1. $u = v_8,\ V_T = \{v_1, v_8\}$
   $V - V_T = \{v_2, v_3, v_4, v_5, v_6, v_7\}$
   $d = (0, 2, \infty, \infty, \infty, \infty, 4, 1)$
2. $u = v_2,\ V_T = \{v_1, v_8, v_2\}$
   $V - V_T = \{v_3, v_4, v_5, v_6, v_7\}$
   $d = (0, 2, 4, \infty, \infty, \infty, 4, 1)$
3. $u = v_3,\ V_T = \{v_1, v_8, v_2, v_3\}$
   $V - V_T = \{v_4, v_5, v_6, v_7\}$
   $d = (0, 2, 4, 5, 5, \infty, 4, 1)$
4. $u = v_7,\ V_T = \{v_1, v_8, v_2, v_3, v_7\}$
   $V - V_T = \{v_4, v_5, v_6\}$
   $d = (0, 2, 4, 5, 5, 8, 4, 1)$

   The remaining iterations do not change d

Dense Graphs

Distribute the adjacency matrix using 1D block distribution, e.g.,



		v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
P_0	v_1		5						1
	v_2	5		2					1
P_1	v_3		2		1	1			
	v_4			1				2	
P_2	v_5			1				6	
	v_6							4	
P_3	v_7				2	6	4		3
	v_8	1	1					3	

Process P_i “owns" a subset V_i of V

P_i computes in the while loop

for all $v \in (V - V_T) \cap V_i$ do

$d[v] = \min\{d[v], d[u] + w(u, v)\}$

- Each P_i needs to know V_T
- P_i finds u such that $d[u] = \min\{d[v] \mid v \in (V - V_T) \cap V_i\}$
  That is, a minimum among the vertices it owns
- P_0 can do global reduction to find u such that $d[u] = \min\{d[v] \mid v \in (V - V_T)\}$
- P_0 broadcasts u
- Each P_i inserts into its V_T

- Let $|V| = n$
- Each process uses $O(n^2/p)$ storage
- Computations are in $O(n^2/p)$
- Reduction and broadcast are in $O(\log p)$
- There are n communication steps
- $T_p = O(n^2/p) + O(n \log p)$
- The speed up is

$$S = \frac{T_s}{T_p} = \frac{O(n^2)}{O(n^2/p) + O(n \log p)} = \frac{p}{1 + O(\frac{p \log p}{n})}$$

- The efficiency is

$$E = \frac{1}{1 + O(\frac{p \log p}{n})}$$

Sparse graphs: Johnson's algoritm

Algorithm (Johnson)

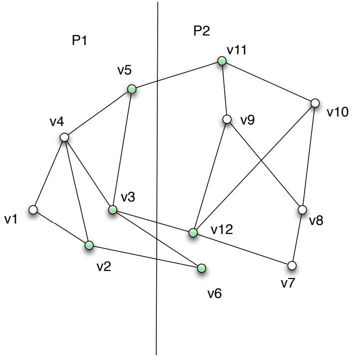
Input: $G = (V, E, w)$, source vertex s
Output: vector d with shortest distances to s
Compute:
% initialization
$Q = V$, s source vertex
for all $v \in Q$ **do** $d[v] = \infty$
$d[s] = 0$
% find distances
while $Q \neq \emptyset$ **do**
  u = extract vertex with smallest distance from Q
  for each $v \in \text{Adj}[u]$ **do**
    if $v \in Q$ and $d[u] + w(u, v) < d[v]$ then
      $d[v] = d[u] + w(u, v)$
      update $d[v]$ in Q

- Q is a priority queue
- Can be implemented as min-heap
- The top of the heap is the vertex with shortest distance
- Distance update (rebuilding the heap) is in $O(\log|V|)$
- For each vertex we scan its incident edges
- $|E|$ edges, $O(\log|V|)$ updates, $O(|E|\log|V|)$ running time
- How to parallelize?

Graph distribution

- Assume p processes
- Partition V into p sets, $V_1, V_2, \ldots, V_p$
- Process i
  - stores V_i
  - stores for each $u \in V_i$ all adjacent vertices to u, i.e., $v \in \mathrm{Adj}[u]$
    and the weights of the corresponding edges
- For $u \in V_i$, let $v \in \mathrm{Adj}[u]$
  - if $v \in V_i$, u is an internal vertex
  - if $v \in V_j$, u is a boundary vertex

Example

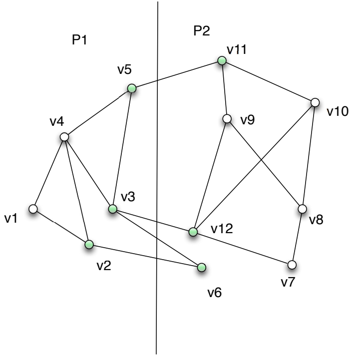


- Process P1 owns
  - internal vertices v_1, v_4
    boundary vertices v_2, v_3, v_5
  - stores vertices from P2: v_6, v_{11}, v_{12}
- Process P2 owns
  - internal vertices v_7, v_8, v_9, v_{10}
    boundary vertices v_6, v_{11},v_{12}
  - stores verices from P1 v_2, v_3, v_5
- Each process also stores the weights of the corresponding edges

Distributed implementation

- Assume p processes
- Partition V into subsets $V_1, \ldots, V_p$
- Process P_i owns V_i and stores adjacent vertices that are on other processes
- P_i maintains its own priority queue Q_i
- P_i keeps an array D, where $D[v]$ is the shortest distance to the source s
  - initially $D[v] = \infty$ for all $v \in V_i$
  - and $D[s] = 0$, where the source is
  - $D[v] = d[v]$, when v is extracted from Q_i
- Each process run JA on its Q_i

- Assume P_i extracts u from Q_i
- If $(u, v) \in E$ and $v \in P_j \neq P_i$, P_i may update $d[v]$ and then send it to P_j
- If v in Q_j, P_j updates Q_j with $d[v]$
- If $v \notin Q_j$, then P_j has already computed shorted distance for v, i.e. $D[v]$
  - if $D[v] \leq d[v] = d[u] + w(u, v)$, there is a longer path, nothing to update
  - if $D[v] > d[v] = d[u] + w(u, v)$, there is a shorter path: insert v back into Q_j



- Each process runs JA
- When u is extracted from Q_i
  if u is boundary, send $d[u]$ to all P_j's that contain a v adjacent to u
- e.g. when P1 updates the distance of v_{12}, send this distance to P2

- Assume s on P_0
- Initially only Q_0 is not empty
- P_0 starts working
- As distances of vertices on other processes become available, priority queues of other processes get populated
- Assume a grid graph, where s is at the bottom left corner
- The computation proceeds like a wave across the grid

2D-block mapping

- Let $|V| = n$ vertices
- Consider a grid of $\sqrt{p} \times \sqrt{p}$ processes
- Assign a block of $n/\sqrt{p} \times n/\sqrt{p}$ vertices to each process
- If s is in the left bottom corner, the wave moves diagonally up the grid of processes
- No more than $O(\sqrt{p})$ processes are busy at any time
- Assume the work of the sequential algorithm is $T_s = W$
- Ignoring communication cost, $T_p = W/\sqrt{p}$
- Speed up is $S = \sqrt{p}$
- Efficiency is $E = 1/\sqrt{p}$

1D-block mapping

- Subdivide the grid vertically in n/p stripes
- Assign each of them to a processor
- On average $p/2$ processes are busy
- $S = p/2$, $E = 1/2$