Paraflow
A Parallel, Maximum Flow Solver

Alejandro Cabrera
cabrera@cs.fsu.edu

Jessie Sanders
jsanders@cs.fsu.edu
Overview

- References
- The Maximum Flow Problem
- Background
- Paraflow Algorithm
- Related Work
- Future Directions
References


Maximum Flow: Definitions

- Flow Network: A directed graph $G=(V,E)$ with each edge having a positive capacity $c_e$, a distinguished source node $s$, and a distinguished sink node $t$.
- Flow: a quantity that is moved through edges in the network.
Maximum Flow: Constraints

- For any given edge $e$:
  - The flow cannot exceed $c_e$

- For any given node $u$:
  - The flow going into $u$ must equal the flow coming out of $u$. 
Maximum Flow: Problem Definition

- Given a network flow graph, find the maximum flow.
- In other words, determine how much flow can be pushed through the network flow graph.
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Discoverer</th>
<th>Running Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dinic's</td>
<td>Dinic (1970)</td>
<td>$O(n^2 m)$</td>
</tr>
<tr>
<td>Karzanov's</td>
<td>Karzanov (1974)</td>
<td>$O(n^3)$</td>
</tr>
<tr>
<td>Shortest Augmenting Path</td>
<td>Ahuja &amp; Orlin (1991)</td>
<td>$O(n^2 m)$</td>
</tr>
<tr>
<td>Capacity-scaling</td>
<td>Gabow (1985)</td>
<td>$O(nm \log U)$</td>
</tr>
<tr>
<td></td>
<td>Ahuja &amp; Orlin (1991)</td>
<td></td>
</tr>
<tr>
<td>Highest-label</td>
<td>Goldberg &amp; Tarjan (1986)</td>
<td>$O(n^2 m^{1/2})$</td>
</tr>
<tr>
<td>FIFO algorithm</td>
<td>Goldberg &amp; Tarjan (1986)</td>
<td>$O(n^3)$</td>
</tr>
<tr>
<td>Lowest Label</td>
<td>Goldberg &amp; Tarjan (1986)</td>
<td>$O(n^2m)$</td>
</tr>
<tr>
<td>Original Excess-scaling</td>
<td>Ahuja &amp; Orlin (1989)</td>
<td>$O(nm + n^2 \log U)$</td>
</tr>
<tr>
<td>Stack-scaling</td>
<td>Ahuja et. al. (1991)</td>
<td>$O(nm + ((n^2 \log U) / (\log \log U)))$</td>
</tr>
<tr>
<td>Wave-scaling</td>
<td>Ahuja et. al. (1991)</td>
<td>$O(nm + n^2 \sqrt{\log U})$</td>
</tr>
</tbody>
</table>
Background: Prolific Research

Theoretical improvements in algorithmic efficiency for network flow problems - ruc.dk (PDF) - Find it @ FSU
ABSTRACT. This paper presents new algorithms for the maximum flow problem, the Hitchcock transportation problem, and the general minimum-cost flow problem. Upper bounds on the numbers of steps in these algorithms are derived, and ...
Cited by 926 - Related articles - BL Direct - All 14 versions

Multiprocessor scheduling with the aid of network flow algorithms - Find it @ FSU
Abstract-In a distributed computing system a modular program must have its modules assigned among the processors so as to avoid excessive interprocessor communication while taking advantage of specific efficiencies of some ...
Cited by 494 - Related articles - View as HTML - All 7 versions

Integer and combinatorial optimization
GL Nemhauser, LA Wolsey, 1999 - ulb.tu-darmstadt.de
... 5.3. Modeling with Binary Variables II: Facility Location. Fixed-Charge Network Flow, and Traveling Salesman 7 ... 469 4. Fixed-Charge Network Flow Problems 495 ...
Cited by 4334 - Related articles - View as HTML - Find it @ FSU - Library Search - All 5 versions

Network flow and testing graph connectivity - Find it @ FSU
NETWORK FLOW AND TESTING GRAPH CONNECTIVITY 509 2. Zero-one network flow. Consider now a network flow problem, as above, except that for all eE, c(e) = 1. (One should realize that even if for all eE, c(e) is integral, not ...
Cited by 318 - Related articles - All 3 versions

Network flow programming
PA Jensen, JW Barnes, 1980 - John Wiley & Sons Inc
Cited by 161 - Related articles - Find it @ FSU - All 2 versions

A network flow computation for project cost curves - Find it @ FSU
DR Fulkerson - Management Science, 1961 - jstor.org
A network flow method is outlined for solving the linear programming problem of computing the least cost curve for a project composed of many individual jobs, where it is assumed that certain jobs must be finished before others ...
Cited by 183 - Related articles - All 5 versions
Background: Prolific Research

Since 2009, Google Scholar is able to locate roughly 24,000 articles and patents related to network flow. Assuming only 1% of all search findings are actually interesting and relevant, that's still 240 articles or patents related to network flow!
Background: Applications of Network Flow

• Matching - Bipartite Matching
• Scheduling - Airline Scheduling
• Cost-Analysis - Project Selection
• Layer Separation - Image Segmentation
• Predicting Winners - Baseball Elimination
• Transportation Optimization – Packet Routing
Paraflow Overview

- Tools Utilized
- Sequential Algorithm
- Heuristics
  - Global
  - Gap
- Parallel Algorithm Modifications
Tools Utilized

- Boost Graph Library
  - Helpful for representing graphs
- Boost Property Map Library
  - Useful for associating properties with a vertex/edge
  - Hides implementation details from user of Paraflow
- OpenMP
  - Shared-memory parallelism
- STL (parallel version available)
  - vectors, lists, and min/max algorithms
- Boost queue
Paraflow

- Paraflow is based off of Golderg's and Tarjan's Highest-Label preflow-push algorithm
- With proper implementation, this guarantees a $O(n^2 m^{1/2})$ run-time complexity
Paraflow: Data Structure Review

• A Graph has:
  • Vertices
  • Edges
  • Layers

• A vertex has:
  • Distance label
  • Excess
  • Color
  • Out-edge adjacency list
Paraflow: Data Structure Review

- An edge has:
  - Source and target vertex
  - Capacity
  - Residual capacity (flow)
  - Reverse edge (to represent residual graph)

- A layer has:
  - Active vertex list
  - Inactive vertex list
Paraflow: Data Structure Review

• Additional graph data:
  • highest vertex distance label
  • maximum active vertex distance label
  • minimum active vertex distance label
Paraflow: Data Structure Semantics

- Distance label – Ideally, the exact distance of a vertex from the sink. Usually, the relative height of a vertex compared to its peers. Calculated using a BFS. May fall out of sink as a result of relabels. A vertex may only push excess to a peer of lower height.

- Color – Used for BFS/DFS operations.
Paraflow: Data Structure Semantics

• Layer
  • Divided into n levels, where n = |V|
  • Maintains an active vertex list and an inactive vertex list at each level
  • Level corresponds to a vertex's distance label
Paraflow: Sequential Algorithm

FlowValue paraflow(G, src, sink)
initialize(G, src, sink)
while(max active label \geq \text{min active label})
  \hspace{1em} u \leftarrow \text{get\_highest\_active\_label\_vertex}(G)
  \hspace{1em} \text{discharge}(u)
return excess at sink
Paraflow: Initialization

```c
void initialize(G, src, sink)
    init_edge_capacity(G)
    push_excess_from_src(G, src)
    init_distances(G, src, sink)
```
Paraflow: Initialization

void init_edge_capacity(G)
  for each vertex $u$
    for each edge $a$ coming out of $u$
      $a_r \leftarrow$ reverse edge of $a$
      residual_capacity($a$) = capacity($a$)
      residual_capacity($a_r$) = capacity($a_r$)
void push_excess_from_src(G, src)
    for each edge a out of src
        aᵣ ← reverse edge of a
        v ← target(a)
        if(v != src)
            delta ← residual_capacity(a)
            residual_capacity(a) -= delta
            residual_capacity(aᵣ) += delta
            excess(v) += delta
void init_distances(G, src, sink)
    for each vertex u in G
        color(u) = white
    color(sink) = gray
    distance(sink) = 0
    distance_reverse_BFS(G, src, sink)
Paraflow: Initialization

void distance_reverse_BFS(G, src, sink)
    q.push(sink)
    while(!q.empty)
        u ← q.pop()
        for each reverse edge $a_r$ out of u
            v ← target($a_r$)
            update_distance(v)
        endif
    endfor
endwhile
void update_distance(v)
    if(color(v) != gray)
        color(v) = gray
        distance(v) = distance(u) + 1
        q.push(v)
    if(excess(v) > 0)
        add_active_vertex(v)
    else
        add_inactive_vertex(v)
    endif
endif
Paraflow: Choosing the Next Vertex

Vertex get_highest_label_active_vertex(G)
    L ← layer[max_active_label]

    repeat this next step until you have a vertex
    or until the max_active_label < min_active_label
    if(L's active vertex list is empty)
        --max_active_label
    u ← first vertex on L's active vertex list
    return u;
Paraflow: Discharging

- Two primitive operations:
  - push(edge)
  - relabel(vertex)
Paraflow: Push

Pre-reqs: e is a residual edge and is admissible

```c
void push(e)
    e_r ← reverse_edge(e)
    u ← source(e)
    v ← target(e)

    delta ← min(residual_capacity(edge), excess(u))
    residual_capacity(e) -= delta
    residual_capacity(e_r) += delta
    excess(u) -= delta
    excess(v) += delta
```
Paraflow: Relabel

Pre-reqs: u could not be pushed

void relabel(u)

    distance(u) = \min_i (e(u,v_i) \mid distance(v_i))

    highest_label = \max(highest_label, distance(u))
Paraflow: Heuristics

- At this point, Paraflow runs and computes a correct maximum flow.
- However, it is **SLOW**.
- Speeding up:
  - Global Relabel Heuristic
  - Gap Relabel Heuristic
Paraflow: Heuristics

- Global Relabel – Periodically recompute the exact distance labels of all vertices in the graph.
  - Performed every $X$ number of steps:
    - We use $X = 2 \times (8n + m)$
  - On each relabel, increment a work counter by $Y$
    - We use $Y = 2$
- When implemented correctly, gave speed-ups between 2.5x and 3.0x.
Paraflow: Heuristics

- Gap Relabel – If a vertex is relabeled to a layer that has no active or inactive vertices, then there must exist a layer gap.
- Layer gap – a group of nodes that have become disconnected from the flow graph.
- Gap relabel prunes these vertices from the graph.
Paraflow: Global Relabel

void global_relabel(G, sink)
    same as reverse_distance_BFS, except replace:
        if(color(v) != gray)
    with
        if(color(v) != gray && is_residual_edge(a_r))

- Consider only edges that could carry flow to the sink in the relabeling
- Can replace reverse_distance_BFS in initialization with global_relabel.
Paraflow: Gap Relabel

```c
void gap_relabel(u)
    gap_layer_level ← distance(u)
    for each layer L between gap_layer_level and highest_label
        for each inactive vertex u in layer L
            distance(u) = n
    highest_label = max_active_label = distance(u) – 1
```
Paraflow: Now Parallel!

- Finding the maximum flow is not an easy problem to parallelize.
- Currently, there are only a handful of papers available that attempt to do so.
  - Two of them are referenced at the start of this presentation.
- However, it is possible. Our approach is detailed in the following slides.
Paraflow: Parallel Algorithm

- Overview
- Modifications
- Work-sharing load balancing
  - Choosing the highest label vertex
- Concurrent gap relabeling
- Concurrent global relabeling
Paraflow: Overview of Parallel Algorithm

- In order to parallelize a preflow-push based network flow algorithm, sources of parallelism had to be located.

- These include:
  - Distributing the work of discharging vertices across multiple PEs.
  - Concurrently running global relabeling to further amortize cost
  - Concurrently running gap relabel to remove unnecessary vertices.
Paraflow: Modifications Needed

• The algorithm maintains a global Layer vector

• Layers ← Queues[3,4]
  • Each thread maintains two local Layer vectors
    – in-layer – Used to take active vertices from the global layer vector
    – out-layer – A fixed size buffer that stores active vertices produced by discharges performed on in-layer vertices.
      • Once full, ship back to global layer
Paraflow: Work-Sharing

- Need one mutable parameter – batch_size
- Rules to alter batch_size:
  - Divide by 2 if at least max(2, 15%) PEs are idle
  - Multiply by 2 if active PEs + active vertices/batch_size > 1.5 * num_PEs
  - Check above rules every 200 discharges
- Batch_size is used to determine the number of vertices a PE takes from the global layer vector when they run empty.
Paraflow: Concurrent Global Relabeling

• Requires modifications to push and to global relabel

• Requires addition of following per-vertex data:
  • wave – number of times a vertex has been globally updated
  • lock – to synchronize vertex accesses
Paraflow: Concurrent Global Relabeling

Pre-reqs: lock(u), lock(v), d(u) = d(v) + 1, wave(u) = wave(v)

void new_push(e)
   push(e)

• The only modification is to the preconditions for enabling a push operation.
Paraflow: Concurrent Global Relabeling

- When accessing a vertex \( v \) in \( e=(u,v) \) during the BFS step, the following must be true:
  - The lock for \( v \) must be held
  - \( \text{wave}(v) < \text{currentWave} \), where \( \text{currentWave} \) is the number of times a global relabel operation has completed for the graph.
Paraflow:
Concurrent Global Relabeling

• One additional problem:
  • One PE may run global update on a vertex v held by another PE
  • When PE returns v to global layer, it will deposit v in the wrong bucket
  • To solve this, maintain a bit vector worked_on with one bit per vertex:
    – If v is globally updated, set worked_on[v] = true
  • Now, when deposit time comes, a PE need only check this vector to ensure correct placement
Paraflow: Concurrent Gap Relabeling

- Requires modification to relabel operation and gap relabel
  - Relabel:
    - If gapFlag is set, set distance(v) to n
    - Else, resume normal operation
  - Gap Relabel:
    - If a vertex belongs to the gap, set gapFlag(v) → true
- Requires addition of per-vertex data:
  - gapFlag – If a vertex has been flagged as belonging to the gap, set this value to true.
Paraflow: Future Work

- Performance bottlenecks
  - Memory-use
    - For the sequential version, must use $6n + 3m$ memory
    - For parallel, $8n + 3m$
    - Compression?
  - Cache-awareness
    - Maintain an edge and its corresponding reverse edge nearby
- I/O
- Design bottlenecks
  - Parser – Forces reverse edges to be stored apart from edges
Paraflow: Future Work

- Graph-class specific benchmarking
  - How does Paraflow perform on:
    - Acyclic dense graphs
    - Acyclic sparse graphs
    - Grid-style graphs
    - Graphs with many loops
    - Purely random graphs
    - Complete graphs

- Distributed implementation
- Improved rules for PE retreating
Thank you!