CS 5220

Graph Algorithms

David Bindel 2024-11-21

1

Logistics

- Nov 26: Async iterations / final outline
- Dec 3 and 5: Project presentations

- Open Dec 2-12
- \cdot Completion counts toward participation

Midnight on 12/19 (must be after 4:30 PM per university)

- Think "outline of my paper"
- Should include
 - Basics of what you want to do
 - Evaluation setup (workload, etc)
 - Planned (or finished) performance experiments
 - Possibly a timeline
- Can represent reconfigured group

- Logistics
 - Think 3-5 minutes
 - Not everyone needs to talk!
 - Slides via Zoom share
- Contents
 - $\cdot\,$ What you want to do / have done
 - Evaluation and performance plans
 - Does not have to be finished!

- Care most about final report
 - I don't want to stare at your code!
- Outline/presentation help me give feedback
- Emphasis is performance analysis and tuning
 - Think about benchmarks and baselines
 - Think about strong/weak scaling experiments
 - $\cdot \,$... and check for correctness as well
- Get something done, fine to speculate on next steps

Lecture

- Some background on graphs
- Applications and building blocks
- Basic parallel graph algorithms
- Representations and performance
- Graphs and LA
- Frameworks

Mathematically: G=(V,E) where $E\subset V\times V$

- $\cdot \;$ Convention: |V|=n and |E|=m
- May be directed or undirected
- May have weights $w_V:V\to \mathbb{R}$ or $w_E:E:\to \mathbb{R}$
- May have other node or edge attributes as well
- $\cdot \; \operatorname{Path}$ is $\left[\left(u_{i}, u_{i+1} \right) \right]_{i=1}^{\ell} \in E^{*}$, sum of weights is length
- + Diameter is $\max_{s,t\in V} d(s,t)$

- + Hypergraph (edges in V^d)
- Multigraph (multiple copies of edges)













Many possible structures:

- Lines and trees
- Completely regular grids
- Planar graphs (no edges need cross)
- Low-dimensional Euclidean
- Power law graphs

• ...

Algorithms are not one-size-fits-all!

| | Planar | Power law |
|---------------|--------------------|--------------------------------|
| Vertex degree | Uniformly small | $P(\deg = k) \sim k^{-\gamma}$ |
| Radius | $\Omega(\sqrt{n})$ | Small |
| Edge sep | $O(\sqrt{n})$ | nothing small |
| Linear solve | Direct OK | Iterative |
| Apps | PDEs | Social networks |

Calls for different methods!

Applications: Routing and shortest paths



Figure 1: image

- Web crawl / traversal
- PageRank, HITS
- Clustering similar documents

Applications: Sparse solvers



Figure 2: image

- Ordering for sparse factorization
- Partitioning
- Coarsening for AMG

Applications: Dimensionality reduction



Figure 3: image

Common building blocks

- Traversals
- Shortest paths
- Spanning tree
- Flow computations
- Topological sort
- Coloring
- ...
- ... and most of sparse linear algebra.

Let $t_p = \operatorname{idealized} \operatorname{time} \operatorname{on} p \operatorname{processors}$

- $\cdot \ t_1 = \mathrm{work}$
- $\cdot t_{\infty} =$ span (or depth, or critical path length)

Don't bother with parallel DFS! Span is $\Omega(n)$.

Let's spend a few minutes on more productive algorithms...

- $\cdot\,$ Push seed node onto queue and mark
- While Q nonempty
 - Pop node from queue
 - Visit node
 - Push unmarked neighbors on queue
 - Mark all neighbors

Simple idea: parallelize across frontiers

- Pro: Simple to think about
- Pro: Lots of parallelism with small radius?
- Con: What if frontiers are small?

Assuming a high-diameter graph:

- + Form set S with start + random nodes, $|S| = \Theta(\sqrt{n}\log n)$
 - $\cdot \,$ long shortest paths go through S w.h.p.
- $\cdot \,$ Take \sqrt{n} steps of BFS from each seed in S
- Form aux graph for distances between seeds
- Run all-pairs shortest path on aux graph

OK, but what if diameter is not large?

- $\cdot \ \operatorname{Set} d[v] = \infty$ for all vertices
- $\cdot \;\; \operatorname{Set} d[s] = 0$ for seed s
- \cdot Until d stops changing
 - $\cdot \;\; {\rm For \; each} \; u \in V$

$$\cdot \ d[u] = \min(d[u], \min_{w \in N(u)} d[w] + 1)$$

Key ideas:

- At some point, switch from top-down expanding frontier ("are you my child?") to bottom-up checking for parents ("are you my parent?")
- Use 2D blocking of adjacency

Classic algorithm: Dijkstra

- Dequeue closest point to frontier, expand frontier
- Update priority queue of distances (in parallel)
- Repeat

Or run serial Dijkstra from different sources for APSP.

Initialize $d[\boldsymbol{u}]$ with distance over-estimates to source

- $\cdot \ d[s] = 0$
- $\cdot \, \operatorname{Repeatedly relax} d[u] := \min_{(v,u) \in E} d[v] + w(v,u)$

Converges (eventually) as long as all nodes visited repeatedly, updates are atomic. If serial sweep in a consistent order, call it Bellman-Ford.

Alternate approach: hybrid algorithm

- Process a "bucket" at a time
- + Relax "light" edges (wt < Δ), might add to bucket
- When bucket empties, relax "heavy" edges a la Dijkstra

- $\cdot \ S \subset V$ independent if none are neighbors.
- Maximal if no others can be added and remain independent.
- Maximum if no other MIS is bigger.
- Maximum is NP-hard; maximal is easy (serial)

- $\cdot \,$ Start with S empty
- + For each $v \in V$ sequentially, add v to S if possible.

- $\cdot \ \operatorname{Init} S := \emptyset$
- $\cdot \ \operatorname{Init} \operatorname{candidates} C := V$
- $\cdot \,\, \mathrm{While}\,\, C \neq \emptyset$
 - Label each v with a random $\boldsymbol{r}(v)$
 - + For each $v \in C$ in parallel, if $r(v) < \min_{\mathcal{N}(v)} r(u)$
 - $\cdot \;$ Move v from C to S
 - $\cdot\,$ Remove neighbors from v to C

Very probably finishes in $O(\log n)$ rounds.

Luby's algorithm (round 1)



Luby's algorithm (round 2)



Many graph ops are

- Computationally cheap (per node or edge)
- Bad for locality

Memory bandwidth as a limiting factor.

Consider:

- 323 million in US (fits in 32-bit int)
- About 350 Facebook friends each
- Compressed sparse row: about 450 GB

Topology (no metadata) on one big cloud node...



Pro: efficient for dense graphs Con: wasteful for sparse case...

- \cdot Tuples: (i, j, w_{ij})
- Pro: Easy to update
- Con: Slow for multiply

- Linked lists of adjacent nodes
- Pro: Still easy to update
- Con: May cost more to store than coord?





Pro: traversal? Con: updates

- Idea: Never materialize a graph data structure
- Key: Provide traversal primitives
- Pro: Explicit rep'n sometimes overkill for one-off graphs?
- Con: Hard to use canned software (except NLA?)

- Really is standard LA
 - Spectral partitioning and clustering
 - PageRank and some other centralities
 - "Laplacian Paradigm" (Spielman, Teng, others...)
- Looks like LA
 - Floyd-Warshall
 - Breadth-first search?

Semirings have \oplus and \otimes s.t.

- $\cdot\,$ Addition is commutative+associative with a 0
- Multiplication is associative with identity 1
- Both are distributive
- $\cdot \ a \otimes 0 = 0 \otimes a = 0$
- But no subtraction or division

Technically modules over semirings

Example: min-plus

- $\cdot \ \oplus = \min$ and additive identity $0 \equiv \infty$
- $\cdot ~\otimes = +$ and multiplicative identity $1 \equiv 0$
- $\cdot \,$ Useful for shortest distance: $d = A \otimes d$

http://www.graphblas.org/

- Version 2.1.0 (final) as of Dec 2023
- \cdot (Opaque) internal sparse matrix data structure
- Allows operations over misc semirings

Several to choose from!

- Pregel, Apache Giraph, Stanford GPS, ...
- GraphLab family
 - GraphLab: Original distributed memory
 - PowerGraph: For "natural" (power law) networks
 - GraphChi: *Chi*huahua shared mem vs distributed
- Outperformed by Galois, Ligra, BlockGRACE, others
- But... programming model was easy
- GraphIt best of both worlds?

- "Think as a vertex"
 - Each vertex updates locally
 - Exchanges messages with neighbors
 - Runtime actually schedules updates/messages
- $\cdot\,$ Message sent at super-step S arrives at S+1
- Looks like BSP

"Scalability! But at what COST?"

McSherry, Isard, Murray, HotOS 15 You can have a second computer once you've shown you know how to use the first one. – Paul Barham (quoted in intro)

- Configuration that Outperforms a Single Thread
- Observation: many systems have unbounded COST!