Direction-Optimizing Breadth-First Search (BFS)
Beamer, et. al., IEEE (2012)

Andrew Feldman
2/8/22

Background

- 2010s - early social media & post-Twitter “era”
- Energized the study of network effects and social networks
- Watts, et. al. (1998) – social networks are “small worlds”
  - Short diameter – “6 degrees of Kevin Bacon”, “Erdos number”
  - Diameter grows as log(n), where n is node-count
- Barabasi, et. al. (1999) – degree distribution follows power law
  - Studied probably P(k) of degree-k nodes in random networks
  - “Scale-free” network – $P(k) \sim k^{-\gamma}$, γ is a statistical parameter of the network
- Kwak, et. al. (2010) – Twitter is “small-word”/ “scale-free”
- BFS is a key primitive for graph analytics

Background – existing work on accelerating BFS

- Agarwal, et. al. (2010) – bitmap caching, communication optimizations for latency-hiding on Intel processors & multi-socket platforms
- Buluc, et. al. (2011) – Partitioning, parallelism and communication
- Hong, et. al. (2011) – frontier-wise hybridization of sequential, multi-core & GPU
- Yoo, et. al. (2005) – partitioning, memory and communication optimization
- Merrill, et. al. (2012) – task & memory management on CPU/GPU
- Chhugani, et. al. (2012) – load-balancing, caching, communication

Motivation

- Prior work –
  - Spot optimization (bitmap)
  - Platform/architecture-driven
  - CPU vs GPU
  - Approaches to parallelism
- What about workload-driven algorithm engineering?
- Beamer, et. al., Direction-optimizing BFS (2012) - algorithm engineering to exploit idiosyncrasies of social networks
  - Especially – “small-world” networks

Motivation – BFS on “small world” networks

BFS – iterative frontier expansion

```
function breadth-first-search(vertices, source)
    frontier ← {source}
    next ← {}
    parents ← [-1,-1,...-1]
    while frontier ≠ {} do
        top-down-step(vertices, frontier, next, parents)
        frontier ← next
        next ← {}
    end while
    return tree
```

Outer loop over frontier levels – limited by diameter

BFS complexity: O(m+n)

- m edges
- n nodes
- **m edges and n nodes will be explored**

Explosion of inner loop over frontier edges -

```
function top-down-step(vertices, frontier, next, parents)
    for v ∈ frontier do
        for n ∈ neighbors[v] do
            if parents[n] = -1 then
                parents[n] ← v
                next ← next ∪ {n}
            end if
        end for
    end for
```

Motivating experiment: top-down BFS

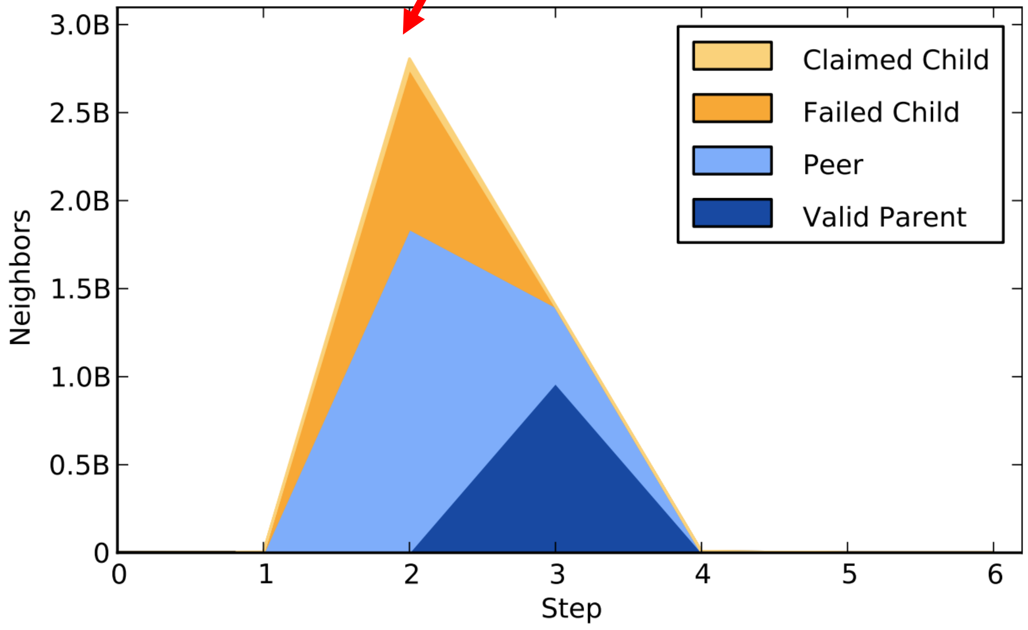
BFS – iterative frontier expansion

```
function breadth-first-search(vertices, source)
    frontier ← {source}
    next ← {}
    parents ← [-1,-1,. . . -1]
    while frontier ≠ {} do
        top-down-step(vertices, frontier, next, parents)
        frontier ← next
        next ← {}
    end while
    return tree
```

"Step" - Outer loop over frontier levels

Inner building next frontier level

Small-world "signature" – most work "squeezed" into first 2-4 frontier iterations



Child node – a node which could be a child of the frontier

Peer – also in frontier

Parent – previous frontier

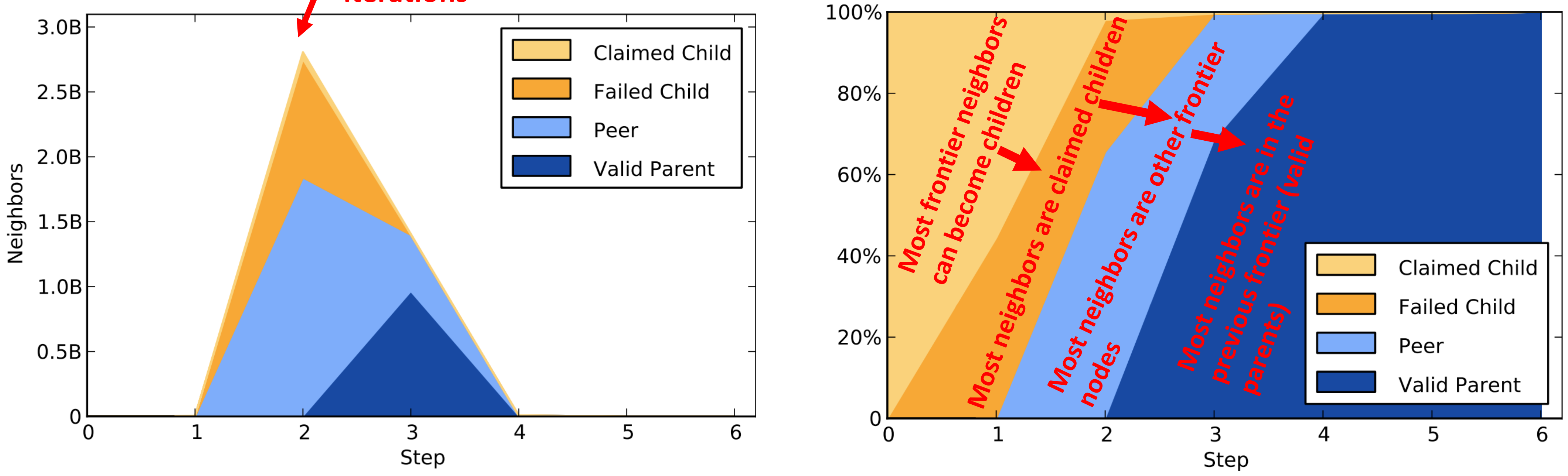
- Absolute breakdown of edge examination work by frontier level (outer-loop step)

Beamer, et. al. *Direction-optimized BFS* (2012)

- Beamer, et. al. contribution: considers the unique impact of low network diameter on BFS
- Naïve BFS is "top-down", however BFS *invariant* is "bottom-up"
  - BFS invariant – every node in the BFS tree must have a parent
- Modify BFS to exploit a unique observation about edge complexity
  - *Theoretical* minimum **edge** examinations is **n – 1** (tree topology)
  - Edges do not need to be explored in both directions
- => "Bottom-up" BFS – minimize edge exploration – iterate over potential *frontierNext* nodes & find first *frontier* neighbor to satisfy invariant
  - Minimizing edge exploration is realistic when most nodes are in *frontier*
  - => Tuning: need to parameterize *which* frontier levels should use "top-down" vs "bottom-up"

Motivating experiment: top-down BFS

Small-world "signature" – most work "squeezed" into first 2-4 frontier iterations



- Absolute & relative breakdown of edge examination work by frontier level (outer-loop step)
- Most edge exploration is unnecessary

Beamer, et. al. Bottom-up BFS

```
BU-BFS (vertices, edges, source)

 while size(frontier) > 0 do:

   parallel-for v in vertices do:
    if parents[v] == -1 do:
     for n in neighbors[v] do:
      if n in frontier do:
       parents[v] <- n
       next <- union(next,v)
       break
      endif
     endfor
    endif
   endfor

 endwhile

endfunction
```

```
parallel-for v in vertices do: //For each node...
 if parents[v] == -1 do: //If invariant is unsatisfied...
  for n in neighbors[v] do: //Search for frontier neighbor..
   if n in frontier do:
    parents[v] <- n //Choose frontier neighbor as parent...
    next <- union(next,v)
    break //Examine no more neighbors!
   endif
  endfor
 endif
endfor
```

Performance comparison

- "Top-down" BFS

```
TD-BFS (vertices, edges, source)

 while size(frontier) > 0 do:
    prefix-sum on degree array (not shown)
    parallel-for v in frontier do:
     parallel-for n in neighbors[v] do:
       if parents[n] == -1 do:
         parents[n] <- v //atomic
         next <- union(next,n)
       endif
     endfor
     filter on frontierNext array (not shown)
    endfor

 endwhile

endfunction
```

Max iterations == graph diameter

Asymptotic sequential complexity:

$O(m+n)$

Worst-case: all nodes

Span: $O(\Delta(\log m))$
Work: $O(m+n)$
Run-time:
$O(m+n + \Delta(\log m))$

Worst-case: all neighbors (serial)

Work-efficient:
Only if $\Delta(\log m)$ is $O(m+n)$

- "Bottom-up" BFS

```
BU-BFS (vertices, edges, source)

while size(frontier) > 0 do:
  parallel-for v in vertices do:
   if parents[v] == -1 do:
    for n in neighbors[v] do:
     if n in frontier do:
      parents[v] <- n
      next <- union(next,v)
      break
     endif
    endfor
   endif
  endfor

endwhile

endfunction
```

Asymptotic sequential complexity:

$O((m+n)\Delta)$

Span: $O(\Delta(\text{mean}_{BFS}(\max_{kernel}(d))))$
Work:
$O((m+n)\Delta)$
Run-time:
$O((m+n)\Delta + \Delta(\text{mean}_{BFS}(\max_{kernel}(d))))$

Work-efficient:
Only if
$\text{mean}_{BFS}(\max_{kernel}(d))$ is $O(m+n)$

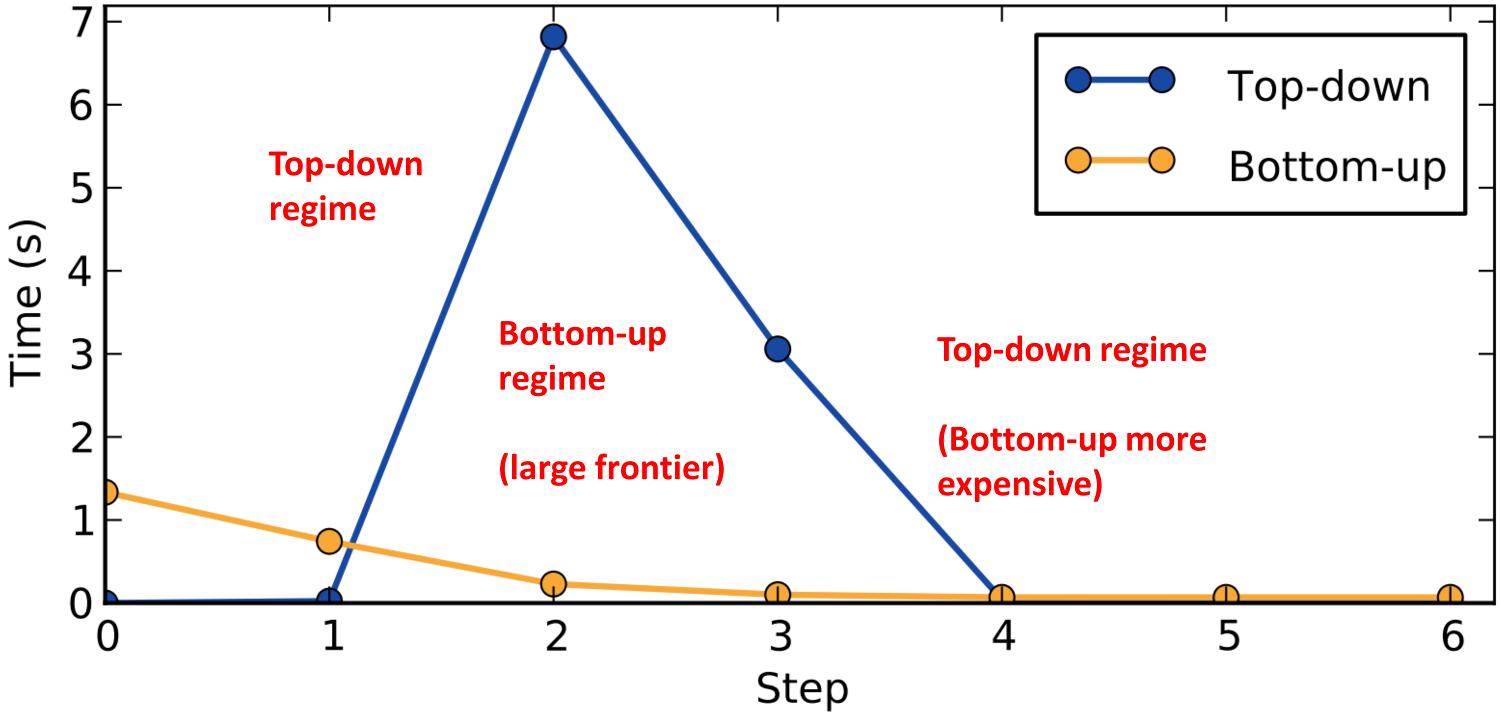
Performance comparison

	TOP-DOWN BFS	BOTTOM-UP BFS
SEQUENTIAL WORST-CASE ASYMPTOTIC PERFORMANCE	$O(m+n)$	$O((m+n)\Delta)$ $O((m+n)\log n)$
PARALLEL WORST-CASE ASYMPTOTIC WORK	$O(m+n)$	$O((m+n)\Delta)$ $O((m+n)\log n)$
PARALLEL WORST-CASE ASYMPTOTIC SPAN	$O(\Delta(\log m))$ $O((\log n)(\log m))$	$O(\Delta(\mathrm{mean}_{BFS}(\max_{kernel}(d))))$ $O((m+n)\log n)$
PARALLEL WORST-CASE ASYMPTOTIC RUN-TIME	$O(m+n + \Delta(\log m))$ $O(m+n)$	$O((m+n)\Delta + \Delta(\mathrm{mean}_{BFS}(\max_{kernel}(d))))$ $O((m+n)(\log n))$
WORK EFFICIENT?	Ambiguous Yes	Ambiguous Yes
BETTER ASYMPTOTIC RUN-TIME, MAKING ASSUMPTIONS ABOUT Δ AND D?	Better	Worse

Above: a comparison of top-down and bottom-up BFS, in terms of asymptotic worst-case sequential and parallel complexity. Values in red assume $\Delta = O(\log n)$ (Watts, et. al. (1998)) and $\mathrm{mean}_{BFS}(\max_{kernel}(d)) = O(m+n)$ or better.

These are asymptotics

Hybrid BFS motivating experiment: Top-down vs bottom-up for BFS

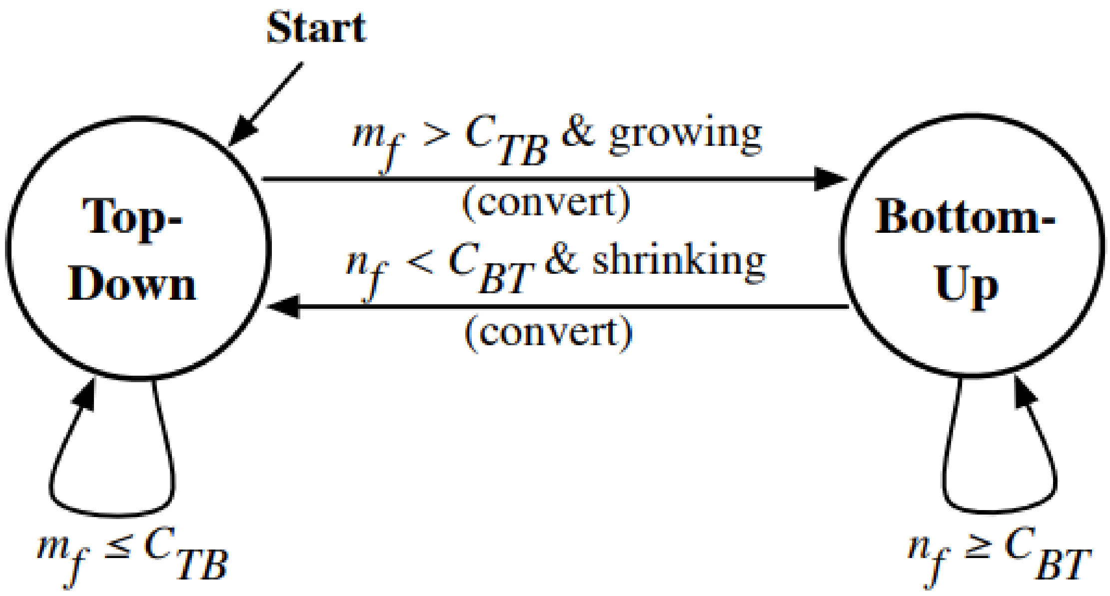


No one-size-fits-all!

Need a way to choose…

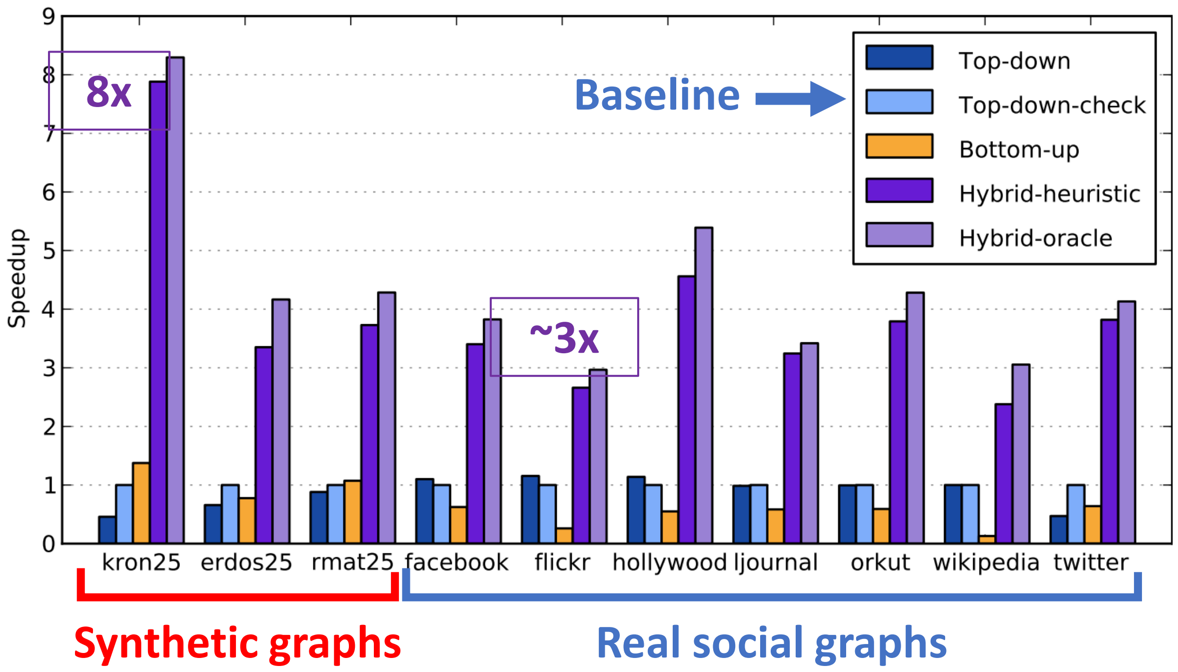
- Bottom-up vs top-down BFS – comparative run-time by frontier level

Hybrid-heuristic BFS



- Heuristic
  - Switch from top-down to bottom-up: $m_f > \frac{m_u}{\alpha} = C_{TB}$ **Large frontier**
  - Switch from bottom-up to top-down: $n_f < \frac{n}{\beta} = C_{BT}$ **Early/late small frontier**
  - Number of edges to check from the frontier (mf)
  - Number of frontier vertices (nf)
  - Number of edges to check from unexplored vertices (mu)

Experiment (real & synthetic data)



- **Speed-up results from avoiding edge examinations**

- **Significant hybrid-BFS speed-up (3x – 8x)**
- **Hybrid-BFS benefit dwarves tuning error - <25% tuning penalty**
- **Bottom-up alone is scarcely more than 1x, sometimes worse**

- **Comparison: Hybrid heuristic, hybrid-oracle, top-down(-check), bottom-up**
- **Workloads: synthetic graphs, real social graphs (Facebook, Flickr, …)**
- Testbench: 16-cores – 2x sockets, each an 8-core (2 thread/core) Intel Sandy Bridge processor @2.7GHz + 20MB LLC, 128GB DRAM

Comparing against prior results

- State-of-the-art prior results
  - Chhugani, et. al. (2012) – load-balancing, caching, communication
  - Hong, et. al. (2011) – frontier-wise hybridization of sequential, multi-core & GPU

Experiment (against prior results)

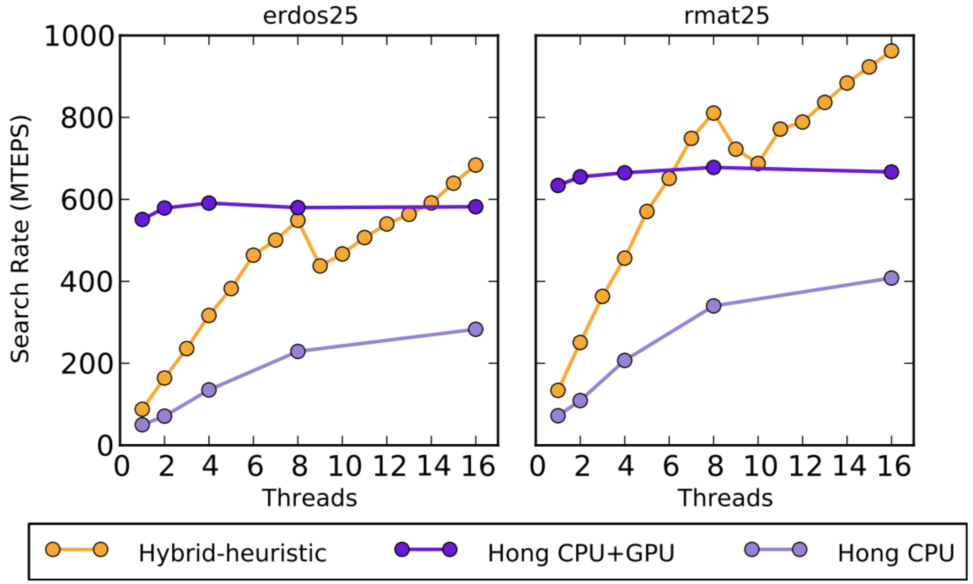


Fig. 16. Search rates on the 8-core system on erdos25 (Uniform Random with 32M vertices and 256M edges) and rmat25 (RMAT with 32M vertices and 256M edges). Other lines from Hong et al. [15]

	rmat-8	rmat-32	erdos-8	erdos-32	orkut	facebook
Prior	750	1100	590	1010	2050	920
8-core	1580	4630	850	2250	4690	1360

TABLE III

PERFORMANCE IN MTEPS OF *Hybrid-heuristic* ON THE 8-CORE SYSTEM COMPARED TO CHHUGANI ET AL. [10]. SYNTHETIC GRAPHS ARE ALL 16M VERTICES, AND THE LAST NUMBER IN THE NAME IS THE DEGREE.

- Comparison against Hong. et. al. and Chhugani et. al
- Testbench: 8-cores – 2x sockets, each an 4-core (2 thread/core) Intel Nehalem-EP processor @2.67GHz + 8MB LLC, 12GB DRAM
- ≥2x speed-up

Comparing against prior results

- State-of-the art GPU result
  - Merrill, et. al. (2012) – task & memory management on CPU/GPU

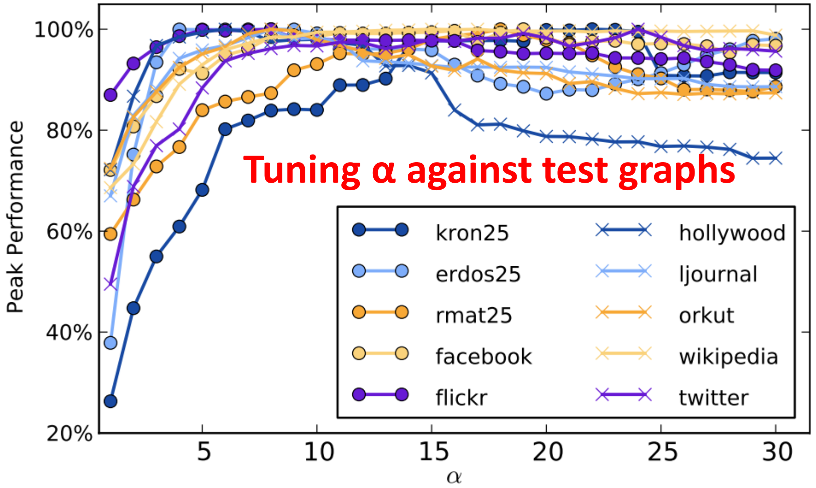
Experiment (against prior results)

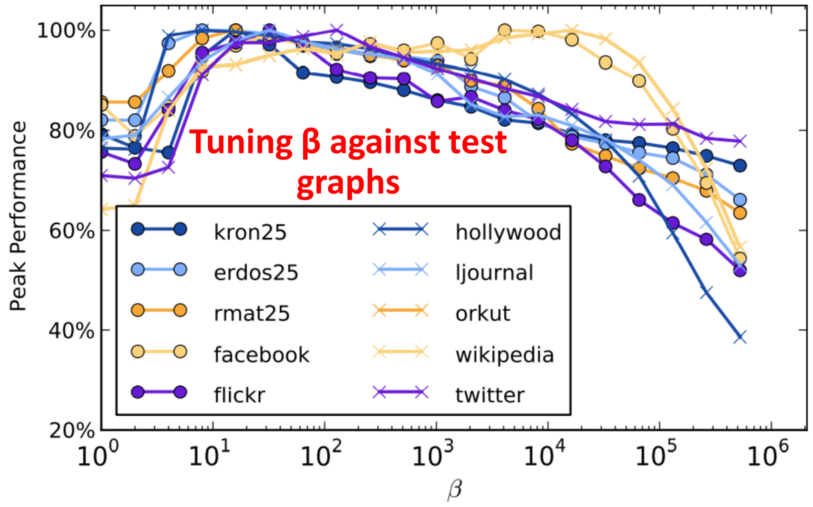
System	kron_ g500-logn20	random. 2Mv.128Me	rmat. 2Mv.128Me
GPU results from Merrill et al. [20]			
Single-GPU	1.25	2.40	2.60
Quad-GPU	3.10	7.40	8.30
Hybrid-heuristic results on multicore			
8-core	7.76	6.75	6.14
16-core	12.38	12.61	10.45
40-core	8.89	9.01	7.14

TABLE IV
Hybrid-heuristic ON MULTICORE SYSTEMS IN THIS STUDY COMPARED TO GPU RESULTS FROM MERRILL ET AL. [20] (IN GTEPS).

- Comparison against Hong. et. al. and Chhugani et. al
- Testbench: 8-cores – 2x sockets, each an 4-core (2 thread/core) Intel Nehalem-EP processor @2.67GHz + 8MB LLC, 12GB DRAM
- ≥2x speed-up

Experiment (parameter tuning)





- Limited sensitivity to parameter tuning

Conclusion

- Strengths – by saving on edge examinations, bottom-up BFS outperforms Top-down BFS *during the frontier step which encompasses most of the graph*.
- One weakness of Bottom-up BFS is that it can be substantially less performant than Top-down BFS,
  - Due to the linear complexity and potentially non-trivial span of the bottom-up BFS kernel
  - Runtime: $O((m+n)\Delta + \Delta(\text{mean}_{BFS}(\text{max}_{kernel}(d))))$

Conclusion

- => Direction-optimized BFS rests heavily on the tuning of the top-down/bottom-up cross-over heuristic
  - The authors demonstrate that hybrid-heuristic BFS performance is not highly-sensitive to tuning parameters
  - Empirically, the hybrid-BFS speed-up dwarves tuning loss, for tuning methodology employed in this work
- Novel result – prior work is focused on spot-optimizations; Beamer, et. al. re-engineering BFS for actually-existing social graph topologies
  - Using application to inform design & experimentation – good AE discipline!
  - Simple & platform independent!

Conclusion

Directions for future work

- Real-world graph degree scaling
- Real-world graph diameter scaling
- Parallelizing the bottom-up “parent search” inner loop
- Maximizing generalizability for tuning parameters
  - Statistical regularization

Discussion questions

- What if the network isn't scale-invariant? What is the impact of scale-invariance on data parallelism
- How would DO BFS handle workloads which do not match its design assumptions?
- Impact of changing workloads?
- How to efficiently model scale-invariance?