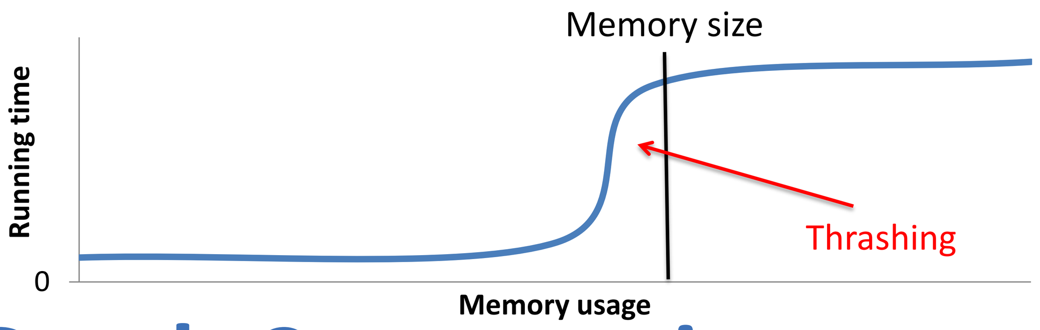
Ligra++: Processing Large Graphs Using Compression

Julian Shun, Laxman Dhulipala, Guy Blelloch
Carnegie Mellon University

Motivation

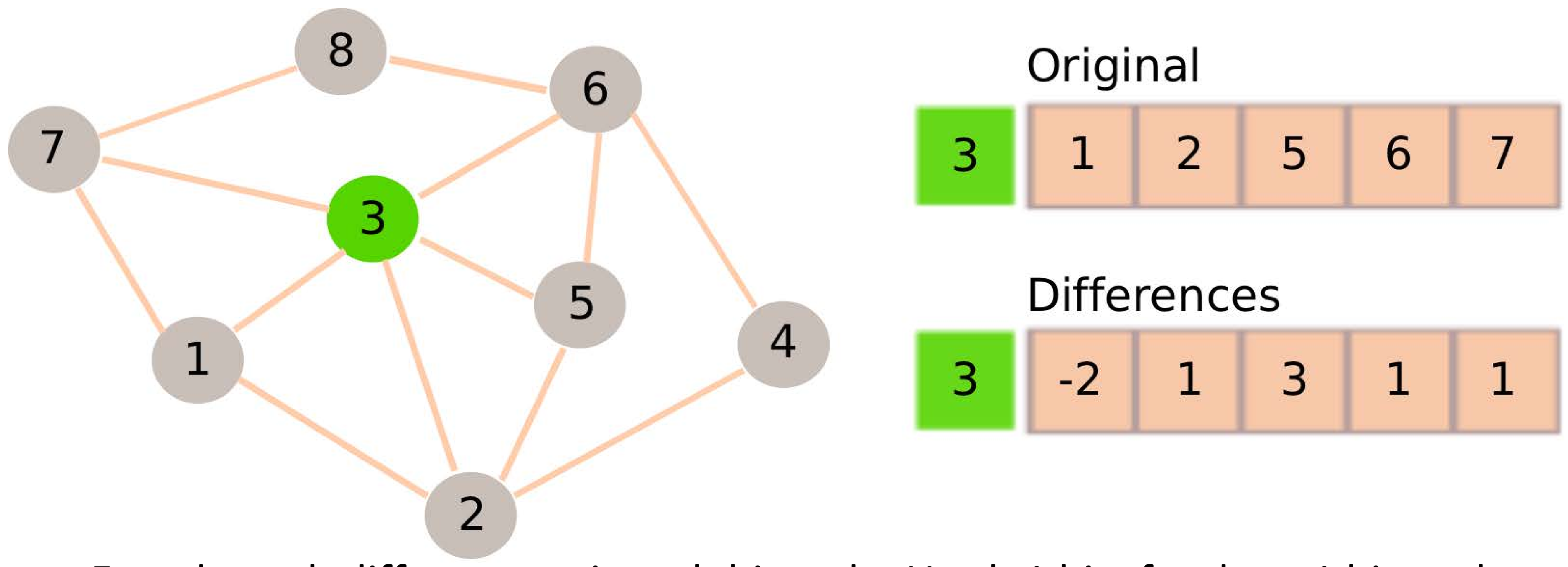
- Growth in graph data sizes (social networks, scientific computing, biology, etc.)
- Need to process graphs quickly
- What approach to use? Distributed memory, shared memory, disk-based
- Shared memory is the fastest, but limited by memory size
- Cost of renting cloud machines increases with RAM size



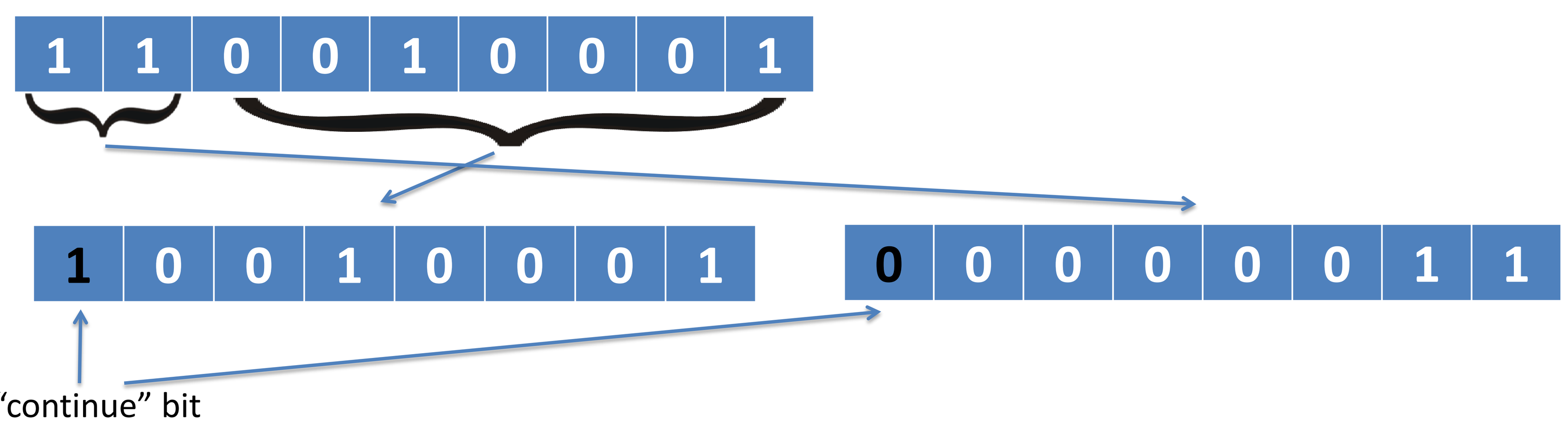
Idea: Use graph compression!

Graph Compression

- Format: for each vertex, store differences between consecutive neighbors



- Encode each difference using a k-bit code. Use k-1 bits for data, 1 bit as the "continue" bit
- We use 8-bit (byte) and 4-bit (nibble) codes
- Example: encode "401" using a byte-code

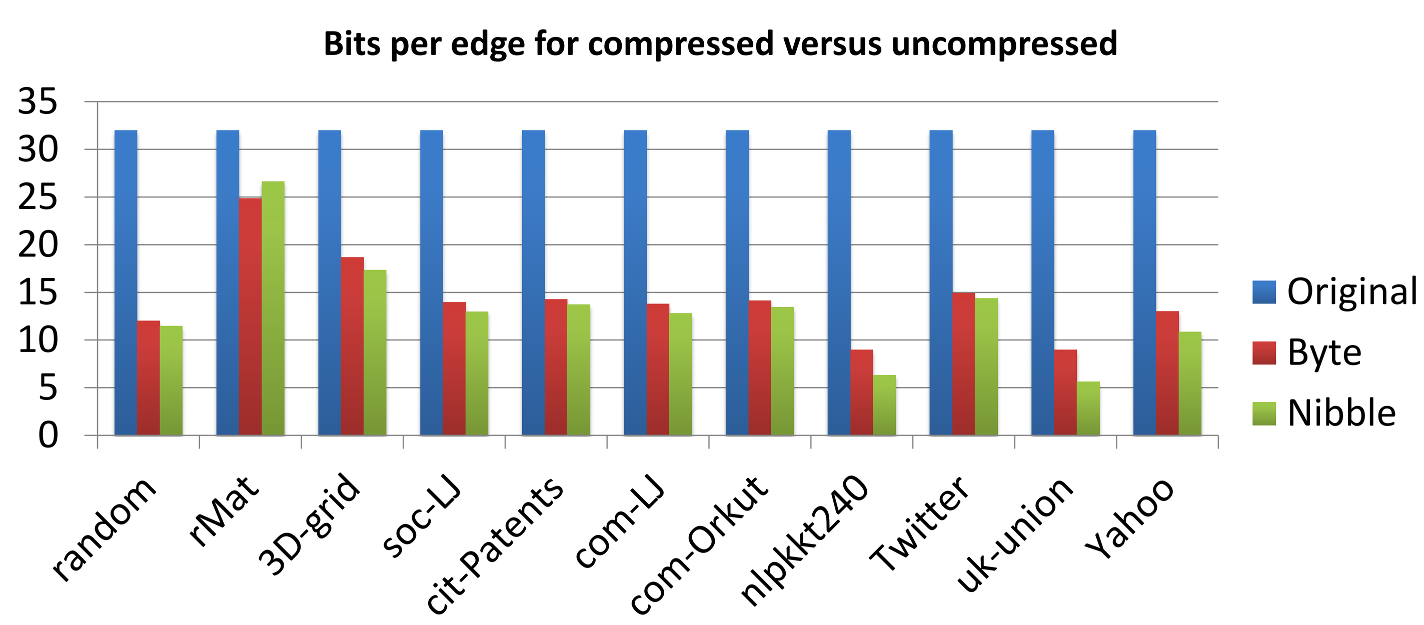


- Note: first difference can be negative, so the first code for it stores a "sign" bit.
- Decoding is just encoding "backwards"

Graph Reordering

- Can run graph reordering ("re-numbering") algorithms to improve locality and compression (and also performance)
- Goal: have neighbors who have ID's close to own ID
- Various reordering algorithms: breadth-first search, depth-first search, hybrid BFS/DFS, METIS (based on finding graph separators), and our own separator-based algorithm
- Using best ordering, we get good compression for most graphs

Input Graph	Ligra
random	433 MB
rMat	465 MB
3D-grid	278 MB
soc-LiveJournal	362 MB
cit-Patents	156 MB
com-LiveJournal	294 MB
com-Orkut	950 MB
nlpkkt240	3.1 GB
Twitter	12.08 GB
uk-union	45.9 GB
Yahoo	62.8 GB



Running times on symmetrized Yahoo graph (1.4 billion vertices, 12.9 billion edges)

40-core Nehalem with hyper-threading	BFS	Betweenness Centrality	Radii	Connected Components	PageRank	Bellman-Ford shortest paths
Original	4.66s	14s	24.5s	12s	8.27s	6.28s
Byte	3.87s	13.1s	23.5s	10.1s	7.47s	9.06s
Nibble	4.85s	18.6s	35.5s	15.7s	9.86s	13.7s

Ligra++

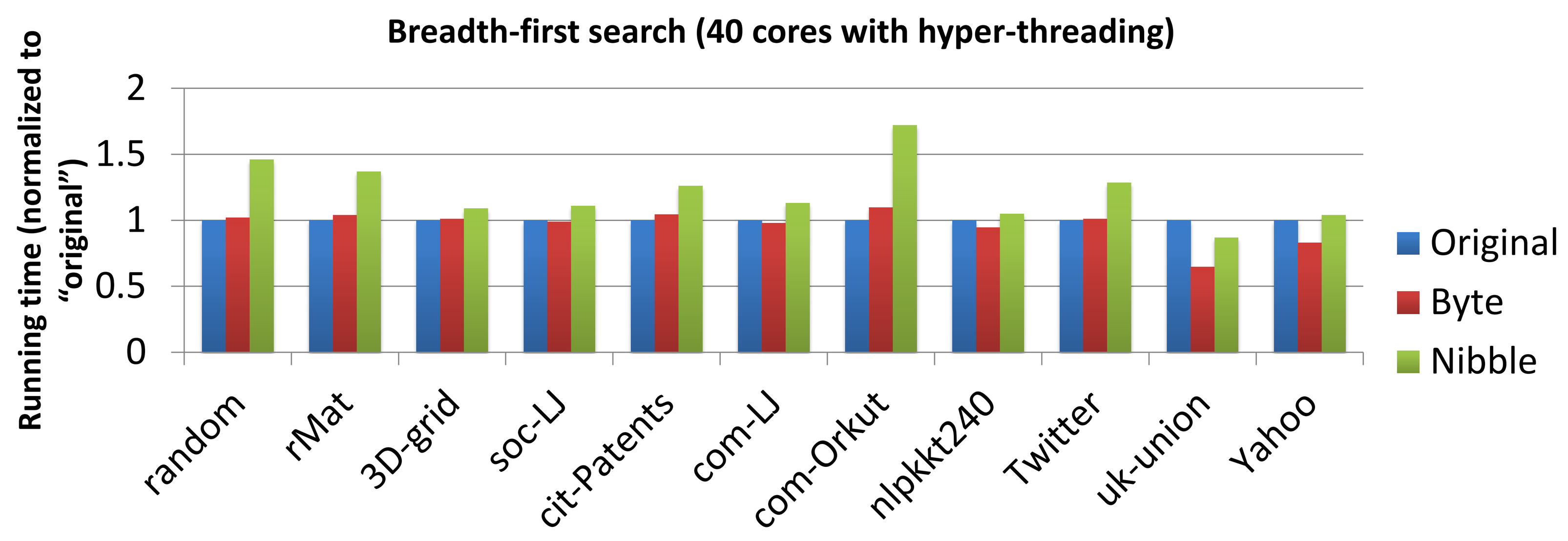
- What about algorithm performance on compressed graphs?
- We implement graph compression and decoding techniques into the Ligra shared-memory graph processing framework

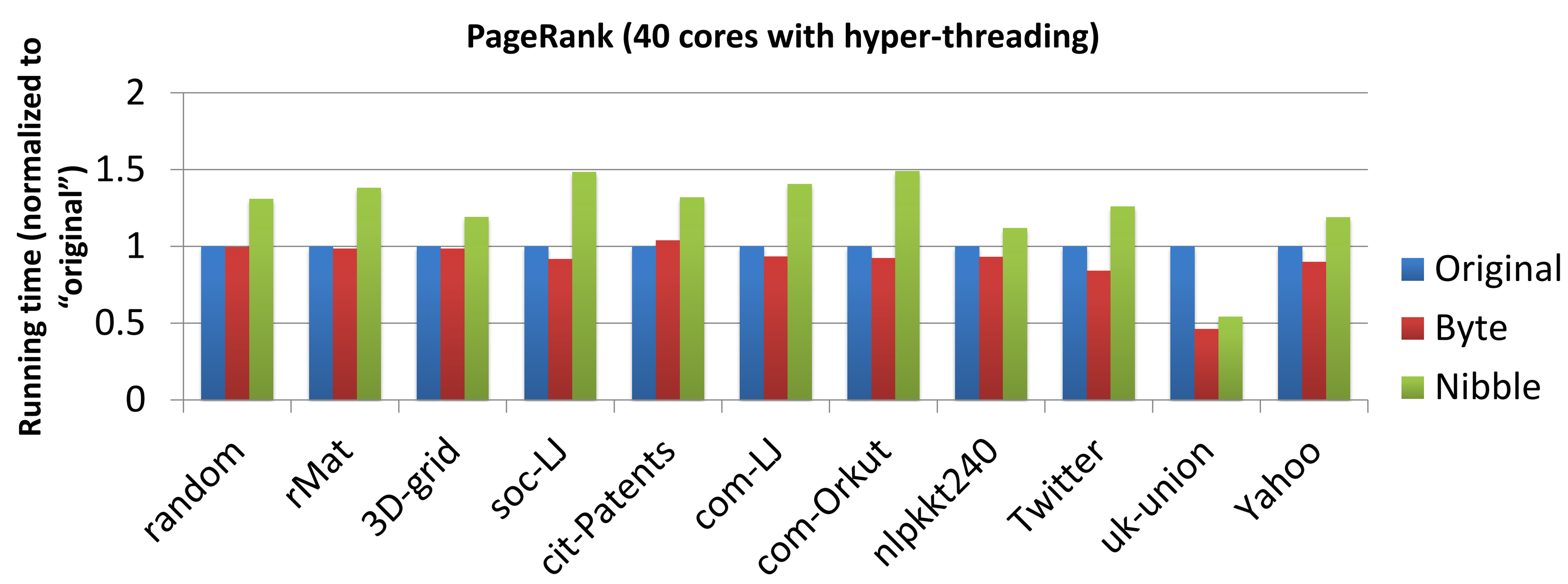
Ligra framework:
represents a subset of vertices in a **vertexSubset**
edgeMap: applies a function to the outgoing edges of a vertexSubset
vertexMap: applies a function to the vertices in a vertexSubset

- We modify the **edgeMap** function to decode each vertex's compressed edges on-the-fly
- To allow for parallel decoding of high-degree vertices, we split the neighbors into chunks, compress each chunk separately, and decode each chunk in parallel
- Encoding cost is amortized across all future computations on the graph

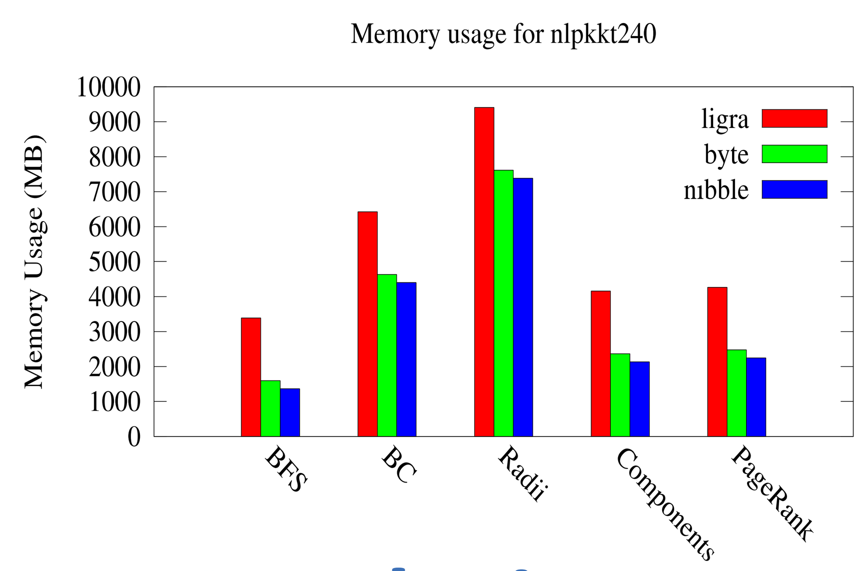
Performance

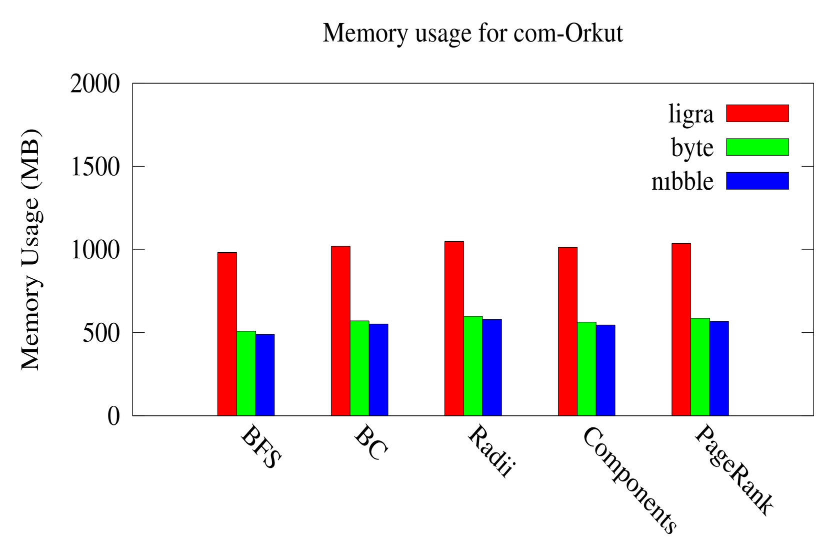
- Trade-offs: compressed versions have smaller memory footprint than uncompressed version, but requires time for decoding
- Performance of compressed versions much better in parallel than sequentially
- In parallel, memory bandwidth/contention is more of a bottleneck, and alleviates the cost of decoding!
- In parallel, byte code performance is competitive with uncompressed version





- Similar trends for other applications: betweenness centrality, radii estimation, connected components, and Bellman-Ford shortest paths
- On 40 cores with hyper-threading, byte codes are between 1.5x slower and 2.7x faster





Conclusions

- With Ligra++, we can fit larger graphs than Ligra with the same amount of memory or the same graph with less memory while maintaining performance
- We are exploring techniques that reduce decoding cost to further improve the running time of Ligra++

Intel Science & Technology Center for Cloud Computing
Carnegie Mellon University · Georgia Tech · intel · Princeton University · UC Berkeley · University of Washington