Gemini: A Computation-Centric
Distributed Graph Processing System

Xiaowei Zhu  Wenguang Chen  Weimin Zheng
Tsinghua University

Xiaosong Ma
Qatar Computing Research Institute

Tsinghua University

Qatar Computing Research Institute

HAMAD BIN KHALIFA UNIVERSITY
Graphs, Platforms, and Systems

Shared-memory
Ligra\textsuperscript{[PPoPP’13]}, Galois\textsuperscript{[SOSP ’13]}
Efficiency
\ding{110}, Scalability
\ding{41}

Distributed
PowerGraph\textsuperscript{[OSDI ’12]}, GraphX\textsuperscript{[OSDI ’14]}
Efficiency
\ding{104}, Scalability
\ding{110}
Connecting Shared-Memory and Distributed

**Profiling existing systems**

<table>
<thead>
<tr>
<th>Nodes System</th>
<th>1 Galois</th>
<th>8 PowerLyra[EuroSys '15]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runtime (s)</td>
<td>19.3</td>
<td>26.9</td>
</tr>
<tr>
<td>Instructions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory references</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Communication (GB)</td>
<td>-</td>
<td>38.1</td>
</tr>
<tr>
<td>Instructions per cycle</td>
<td>0.414</td>
<td>0.655</td>
</tr>
<tr>
<td>L3 cache miss rate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPU utilization</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Far from saturated for IB EDR (100Gbps) (38.1*8/2/26.9/8 = 0.708 Gbps)

- More instructions and memory traffic
- Poorer access locality
- Lower multi-core utilization

Common design

**Bottleneck in computation!**

20 iterations of PageRank on twitter-2010 (41.7M vertices, 1.47B edges)
We Propose: Gemini

- **Build scalability** on top of **efficiency**
  - Avoid unnecessary “distributed” side-effects
  - Optimize computation on partitioned sub-graphs

- **Shift of design focus**
  - Designed for distributed, but **computation-centric**
    - Modern clusters have fast interconnects
    - Computation-communication overlap in place

- **Major optimizations**
  - **Efficiency**
    - Adaptive push-/pull-style computation
    - Hierarchical chunk-based partitioning
  - **Scalability**
    - Locality-aware chunking
    - Chunk-based work-stealing
Dual Mode: BFS Example (1)

|Active edge set| / |E| < threshold
⇒ Sparse mode
⇒ Push operations

Active edge set

Active vertex set

Selective scheduling: only access out-edges from active vertices

1st iteration

Locks/atomic operations required for correctness of concurrent updates
Dual Mode: BFS Example (2)

Active edge set

$|\text{Active edge set}| / |E| > \text{threshold}$

$\Rightarrow \text{Dense mode}$

$\Rightarrow \text{Pull operations}$

2nd iteration

Vertices pulling along in-edges

Contention-free updating

Useless computation
Push vs. Pull: Performance Impact

Single-Source Shortest Paths

- Blue line: Sparse
- Red line: Dense
Distributed Dual-Mode Computation
When Distributed to 2 Nodes

Inter-node message passing

Master

Mirror

Node_0

Node_1
Gemini’s Distributed Push

Masters message mirrors, who update their local neighbors.
Gemini’s Distributed Pull

Mirrors pull updates from neighbors, then message masters
Gemini’s Choice of Graph Partitioning

• Chunking
  – Divide vertex set $V$ into $p$ contiguous chunks
  – Dual-mode edge data distributed accordingly
Why Chunk-Based Partitioning?

• It preserves locality!
  – Fact: **locality** exists in many real-world graphs
    • Vertices “semantically” ordered

  – Preprocessing affordable when vertices unordered
    • E.g., BFS[^Algorithms 09], LLP[^WWW ’11]

---

[^Algorithms 09]: Algorithms 09
[^WWW ’11]: WWW ’11

---

1 The Anatomy of the Facebook Social Graph
More Benefits of Chunking

- Low-overhead distributed designs

- Applied recursively at different levels
  - More on this later
Challenges with Distributed Chunking

• When scaling out
  – CSR/CSC vertex indices can become very sparse
  – Solution: compress vertex indices

• Modern servers built upon NUMA architecture
  – Interleaved layout sub-optimal
  – Solution: apply inter-socket sub-partitioning

• Chunking as vertex-centric (edge-cut) scheme
  – Vertex-centric solutions are not good at load balancing natural graphs
  – Solution: balancing workload in locality-aware manner
Locality-Aware Chunking

- Gemini considers both vertex and edge
  - Edge: the amount of work to be processed
  - Vertex: the processing speed of work (locality)
  - Hybrid metric: $\alpha \cdot |V_i| + |E_i|$

Balancing by edges?

Chunk size affects random access efficiency!
Chunking, Chunking All the Way

Per-node partition

Per-socket partition w. NUMA-aware placement

Data partitioning till socket-level

Per-core work chunk

Mini-chunk for work-stealing

Work partitioning within socket

Locality-aware

Fine grain (64-vertex)
Evaluation

• Platform: 8-node cluster
  - Intel Xeon E5-2670 v3 (12-core CPU), 30MB L3 cache
  - 2 sockets sharing 128 GB RAM (DDR4 2133MHz)
  - Network: Mellanox Infiniband EDR 100Gbps

• Applications
  - PageRank (PR) (20 iterations)
  - Connected Components (CC)
  - Single-Source Shortest Paths (SSSP)
  - Breadth-First Search (BFS)
  - Betweenness Centrality (BC)

• Input graphs

| Graph     | |V|   | |E|            |
|-----------|---------------|---------------|
| enwiki-2013 | 4,206,785    | 101,355,853   |
| twitter-2010 | 41,652,330   | 1,468,365,182 |
| uk-2007-05  | 105,896,555  | 3,738,733,648 |
| weibo-2013  | 72,393,453   | 6,431,150,494 |
| clueweb-12  | 978,048,098  | 42,574,107,469 |
### Single-Node Efficiency

<table>
<thead>
<tr>
<th>Application</th>
<th>Ligra</th>
<th>Galois</th>
<th>Gemini</th>
</tr>
</thead>
<tbody>
<tr>
<td>PR</td>
<td>21.2</td>
<td>19.3</td>
<td>12.7</td>
</tr>
<tr>
<td>CC</td>
<td>6.51</td>
<td>3.59*</td>
<td>4.93</td>
</tr>
<tr>
<td>SSSP</td>
<td>2.81</td>
<td>3.33</td>
<td>3.29</td>
</tr>
<tr>
<td>BFS</td>
<td>0.347</td>
<td>0.528</td>
<td>0.468</td>
</tr>
<tr>
<td>BC</td>
<td>2.45</td>
<td>3.94*</td>
<td>1.88</td>
</tr>
</tbody>
</table>

*Runtime in seconds (twitter-2010)*

<table>
<thead>
<tr>
<th>System</th>
<th>Ligra</th>
<th>Gemini</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remote access ratio</td>
<td>50.1%</td>
<td>9.10%</td>
</tr>
<tr>
<td>L3 cache miss rate</td>
<td>52.6%</td>
<td>40.1%</td>
</tr>
<tr>
<td>Average access latency</td>
<td>183ns</td>
<td>125ns</td>
</tr>
</tbody>
</table>

**Memory performance (BC)**

More iterations

More instructions

NUMA-aware memory accesses

“*” uses different algorithms.
Multi-Node Scalability: Larger Graphs

| Graph         | |V|   | |E|   |
|---------------|------------------|------------------|
| enwiki-2013   | 4,206,785        | 101,355,853      |
| twitter-2010  | 41,652,330       | 1,468,365,182    |
| uk-2007-05    | 105,896,555      | 3,738,733,648    |
| weibo-2013    | 72,393,453       | 6,431,150,494    |
| clueweb-12    | 978,048,098      | 42,574,107,469   |

318GB input graph

<table>
<thead>
<tr>
<th>Application</th>
<th>PowerLyra</th>
<th>Gemini</th>
</tr>
</thead>
<tbody>
<tr>
<td>PR</td>
<td></td>
<td>31.1</td>
</tr>
<tr>
<td>CC</td>
<td></td>
<td>25.7</td>
</tr>
<tr>
<td>SSSP</td>
<td>Out of memory</td>
<td>56.9</td>
</tr>
<tr>
<td>BFS</td>
<td></td>
<td>10.2</td>
</tr>
<tr>
<td>BC</td>
<td></td>
<td>45.3</td>
</tr>
</tbody>
</table>

Runtime in seconds (clueweb-12)
Multi-Node Scalability: Faster Speeds

**PageRank**
- uk-2007-05: 39.8X
- weibo-2013: 11.3X

**Connected Components**
- uk-2007-05: 36.0X
- weibo-2013: 21.6X
Closing Remarks

- What have we learned
  - Computation efficiency highlighted by fast network
    - Existing guidelines may not apply
- Chunking works!
  - Multi-fold benefits
  - Enables series of optimizations

Search GeminiGraph on Github
Thanks!

Q & A