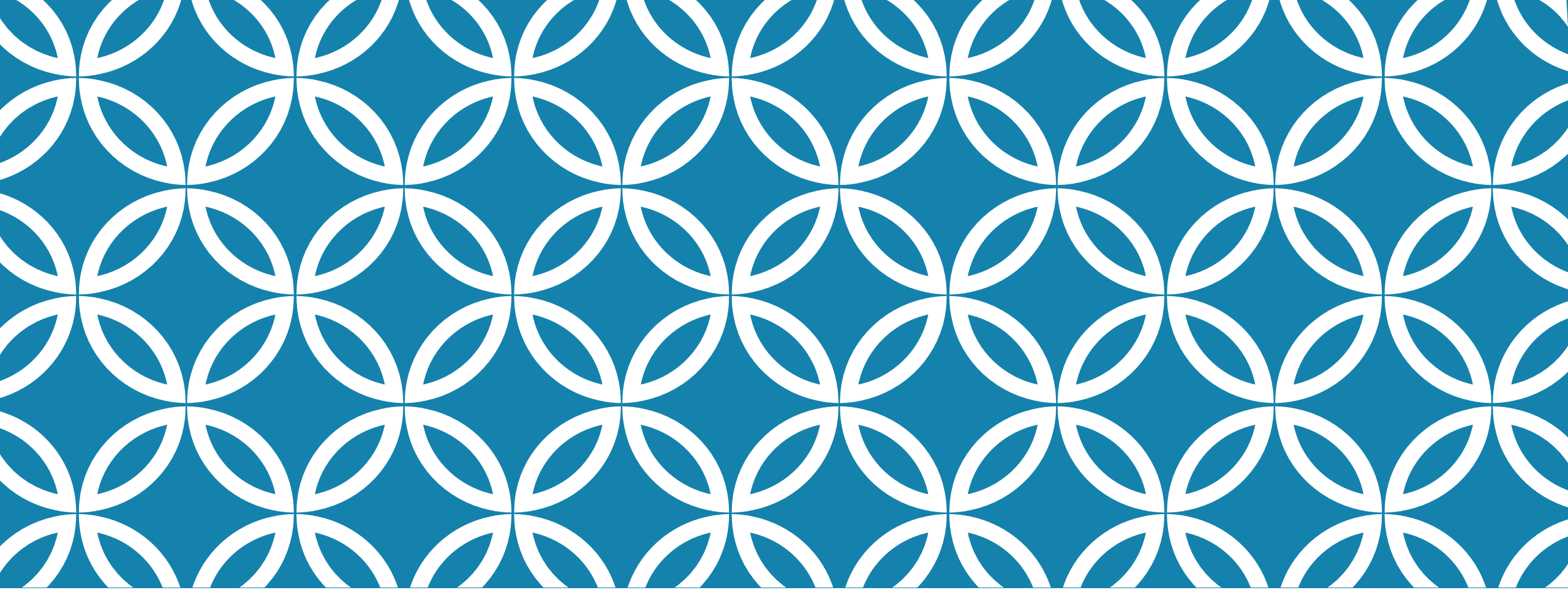


LUMOS: DEPENDENCY-DRIVEN DISK-BASED GRAPH PROCESSING

Endrias Kahssay

MOTIVATION

Disk based processing allows graph processing to scale beyond available memory.

Based on partitioning graph and streaming partitions.

I/O intensive.

MOTIVATION

Want strong consistency (Bulk Synchronous Parallel (BSP)) for the programmer.

Current approaches with BSP optimized for min I/O per iteration, but not globally.

Want faster performance!

ENTER LUMOS

First out of order execution with synchronous semantics.

Out of order is used to reduce I/O by computing multiple iterations of a vertex value at once.

1.8x faster than state of art: Grid Graph.

BACKGROUND: BSP

Synchronous stepwise execution semantics.

Computation in iterations on vertices, and serialization between iterations.

Makes it easy for the programmer to reason about.

Algorithm 1 Synchronous PageRank

1: $G = (V, E)$ ▷ Input graph
2: $\texttt{pr} = \{1, 1, ..., 1\}$ ▷ Floating-point array of size $|V|$
3: **while** not converged **do**
4: $\texttt{newPr} = \{0, 0, ..., 0\}$ ▷ Floating-point array of size $|V|$
5: **par-for** $(u, v) \in E$ **do**
6: ATOMICADD(&$\texttt{newPr}[v]$, $\frac{\texttt{pr}[u]}{|out\_neighbors(u)|}$)
7: **end par-for**
8: **par-for** $v \in V$ **do**
9: $\texttt{newPr}[v] = 0.15 + 0.85 \times \texttt{newPr}[v]$
10: **end par-for**
11: SWAP(pr, newPr)
12: **end while**

PARTITIONING

In disk-based graph processing, graph is too large to fit in memory.

Vertices are separated into P partitions, and each partition holds incoming edges to the vertices inside it.

Iterative engine streams through partitions in fixed order, and then does computation across vertex values. Repeat until convergence.

RELATED WORK

Synchronous Out-of-Core Graph Processing

- GraphChi, X-Stream, GridGraph
- Iterations are serialized
- GridGraph is state of art in performance, due to its mechanism for graph storage on disk

Asynchronous Out-of-Core Graph Processing

- CLIP and Wonderland
- Designed for asynchronous algorithms like SSSP.
- Violate BSP

LUMOS INSIGHTS

- Synchronous Graph Processing is stricter than BSP, losing performance.
- Future value computation: compute values of vertices when their constraints are satisfied.
- No serialization between iterations, and explicitly satisfy the BSP constraint.

SYNCHRONOUS DEPENDENCIES

Vertex value in a given iteration depends on incoming neighbors value from previous iteration.

$$\forall (u,v) \in E, \quad u^t \mapsto v^{t+1}$$

A static graph property

No direct dependence between vertices that are not connected.

PROPGATION

First idea: compute values for nodes for iteration t+1 if all its incoming neighbors value from iteration t are available.

In partition processing, values for vertices in partition P are available when P is processed.

Promising direction, but only works for low-in-degree vertices (only 1-4% edge saving in their experiment)

RELAX!

We can relax the precondition so that we don't need all incoming neighbors to be ready.

As long we use an associative aggregator operator for combining neighbors values, we can keep a partial aggregate for the next time step.

$$v^t = f(\bigoplus_{\forall e=(u,v)\in E} (u^{t-1})) \quad \text{and} \quad g(v^{t+1}) = \bigoplus_{\substack{\forall e=(u,v)\in E \\ s.t.\ p(u)<p(v)}} (u^t) \quad (2)$$

PROGATION

40-50% of values successfully propagated in experiments.

Edges corresponding to propagations don’t need to be loaded in the next time step.

HOW FAR?

Define D as the max distance of Propagation (max difference between current iteration and iteration number for any of the nodes).

Traditional systems achieve : 1. LUMOS: > 1

Propagations drastically decrease after 2, so LUMOS sticks with 2 (iteration t and t+1).

GRAPH LAYOUT

Value propagations can be statically determined because it's a property of the underlying graph.

Create separate graph layouts to avoid reading edges whose dependency is already satisfied.

For D=2, we create a primary and secondary layout.

INTRA-PARTITION PROPAGATION

Natural graphs possess locality due to graph collection strategies / numbering => chunk based numbering has locality.

Partitions have large number of edges that don't cross partitions.

Propagate this internally by buffering the partition in memory!

$$g(v^{t+1}) = \bigoplus_{\substack{\forall e=(u,v)\in E \\ s.t.\ p(u)\leq p(v)}} (u^t)$$

LUMOS SYSTEM

Built on top of Gridgraph, which is a state of the art out-of-core processing system.

Three key programming interfaces:

ProcessPrimary

ProcessSecondary

VertexMap

Algorithm 3 PageRank Example

```
1:  function PROPAGATE(e)
2:      ATOMICADD(&sum[e.target], pagerank[e.source] / outdegree[e.source])
3:  end function
4:  function CROSSPROPAGATE(e)
5:      ATOMICADD(&secondary_sum[e.target],
                  sum[e.source] / outdegree[e.source])
6:  end function
7:  function COMPUTE(v)
8:      sum[v] = 0.15 + 0.85 × sum[v]
9:  end function
10: function ADVANCE(v)
11:     diff = |pagerank[v] − sum[v]|
12:     pagerank[v] = sum[v]
13:     sum[v] = secondary_sum[v]
14:     secondary_sum[v] = 0
15:     return diff
16: end function
17: pagerank = [1, ..., 1]
18: sum = [0, ..., 0]
19: secondary_sum = [0, ..., 0]
20: iteration = 0
21: converged = false
22: while ¬converged do
23:     if iteration % 2 == 0 then
24:         PROCESSPRIMARY(PROPAGATE, CROSSPROPAGATE,
                           COMPUTE);
25:     else
26:         PROCESSSECONDARY(PROPAGATE, COMPUTE);
27:     end if
28:     d = VERTEXMAP(ADVANCE);
29:     converged = d / |V| ≤ threshold
30:     iteration = iteration + 1
31: end while
```

I/O ANALYSIS

$$C = \begin{cases} (1-\alpha) \times |E| \; + \; k \times |V| & \ldots \text{ secondary layout} \\ |E| \; + \; k \times |V| & \ldots \text{ primary layout} \end{cases}$$

Algorithm alternates between primary and secondary layout. Alpha is typically over 0.7.

GRAPH LAYOUT / PARTITIONING

Objective:

$$\arg\max_{\mathscr{P}} |\{(u,v) : (u,v) \in E \ \wedge \ p(u), p(v) \in \mathscr{P} \ \wedge \ p(u) < p(v)\}|$$

STRATEGIES

(A) Highest Out-Degree First: $\forall u, v \in V$,
if $p(u) < p(v)$, then $out\_degree(u) \geq out\_degree(v)$

(B) Highest In-Degree Last: $\forall u, v \in V$,
if $p(u) < p(v)$, then $in\_degree(v) \geq in\_degree(u)$

(C) Highest Out-Deg. to In-Deg. Ratio First: $\forall u, v \in V$,
if $p(u) < p(v)$, then $\frac{out\_degree(u)}{in\_degree(u)} \geq \frac{out\_degree(v)}{in\_degree(v)}$

EXPERIMENTS

Performance tests: run on h1.x2 large with 8 vCPUs, 32GB memory, 2TB HDD.

- Disk speed: 278 MB/sec, memory: 9.6GB/sec

I/O scaling: run on a machine disk bandwidth of 195 MB to 768 MB/sec, and another with 1.2GB/sec to 3.9 GB/sec.

PERFORMANCE

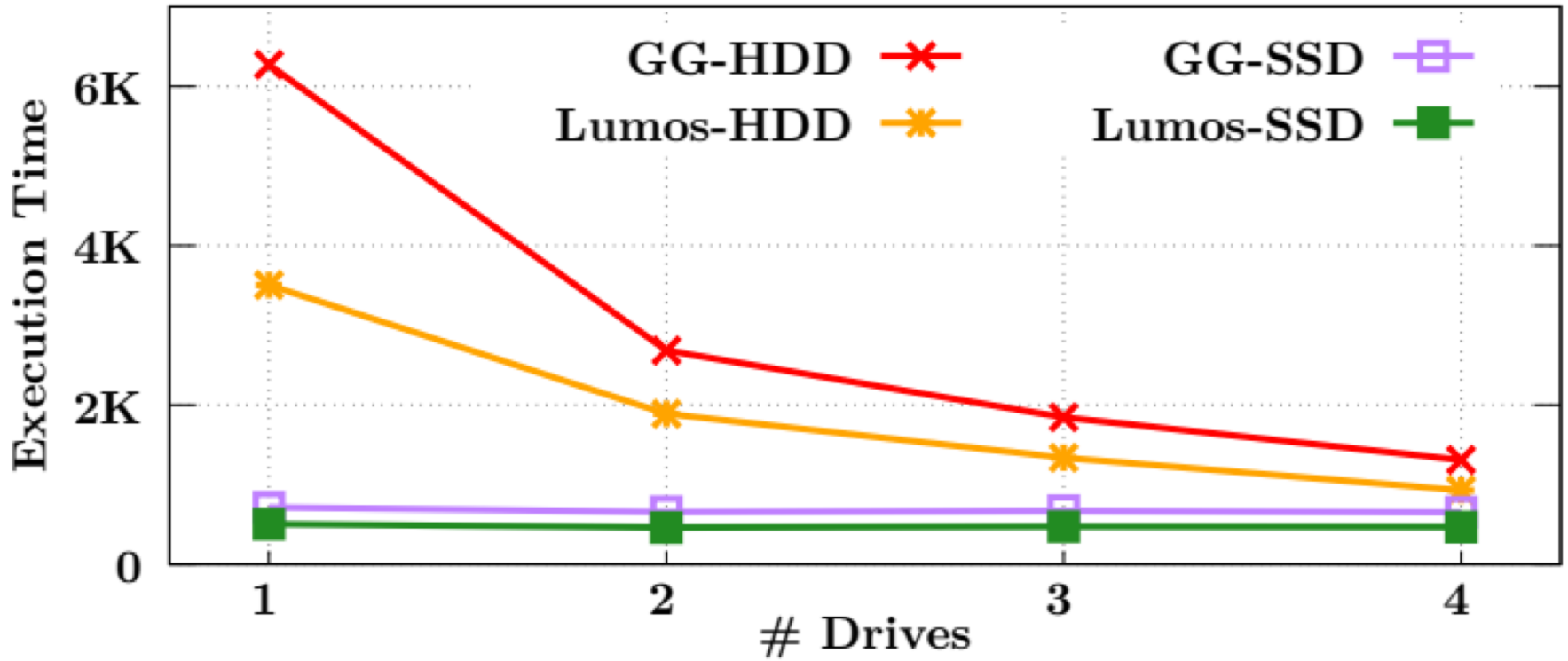
	Version	TT	FT	YH
PR	GridGraph	737	1008	3223
	LUMOS-BASE	563	659	2027
	LUMOS	439	583	1885
	× LUMOS	**1.68×**	**1.73×**	**1.71×**
CoEM	GridGraph	1119	1554	5082
	LUMOS-BASE	861	1029	3216
	LUMOS	651	914	3043
	× LUMOS	**1.72×**	**1.70×**	**1.67×**
DP	GridGraph	846	1032	3484
	LUMOS-BASE	656	675	2219
	LUMOS	498	611	2111
	× LUMOS	**1.70×**	**1.69×**	**1.65×**

TwitterMPI (TT) ~15 – 45GB on disk, 2.0B edges

Friendster (FT), ~20- 60 GB on disk, 2.5B edges

Yahoo (YH), ~50 -150 GB on disk, 6.6B edges

BANDWIDTH TEST



GRAPH PARTITION STRATEGY

Highest out in degree and highest in degree achieves 13 – 89% propagations.

Highest Deg. Ratio First achieves 51-88% propagations.

CONCLUSION

LUMOS is an out of order graph framework that provides synchronous processing guarantees.

Future propagation allows it to reduce I/O misses by reducing the number of edges that are loaded in half of the iterations.

It computes future values across 71-97% of edges, and accelerates out-of-core graph processing by up to 1.8x.

STRENGTHS AND WEAKNESSES

Well organized paper, and easy to understand.

Experiments don't compare against other graph frameworks / and more obsecure graph algorithms.

Graph partitioning was relatively simple, would have been interesting to see more complex approaches.

QUESTIONS

What other heuristics might be interesting for graph partitioning given the objective function?

Is there a way to avoid storing D layouts, and still get the I/O benefits?

How does it compare with other asynchronous graph frameworks?