X-Stream

Pranav Talluri

Background: Graph Processing Problems

- Weakly/Strongly Connected Components
- Single-Source Shortest Paths
- Minimum Cost Spanning Tree
- Maximal Independent Set
- Conductance
- PageRank
- Alternating Least Squares
- Bipartite Matching
- Clustering
- …

- Social Networks
- Internet (Search)
- Telecommunications
- Recommendation Systems
- Transport Networks
- IoT
- Epidemiology

Background: Graph Processing Systems

Partitioning	Edge-Cut	Vertex-Cut	Hybrid-Cut	Stream-Based	Distributed	Dynamic
Unit	Vertex-Centric	Edge-Centric				
Model	Superstep	Scatter-Gather	Gather-Apply-Scatter (GAS)	Frontier		
Dynamism	Static	Temporal	Streaming			
Workload	Local	Global				
Computing	Single-Machine	Distributed				

Background: Pregel

Partitioning	Edge-Cut	Vertex-Cut	Hybrid-Cut	Stream-Based	Distributed	Dynamic
Unit	Vertex-Centric	Edge-Centric				
Model	Superstep	Scatter-Gather	Gather-Apply-Scatter (GAS)	Frontier		
Dynamism	Static	Temporal	Streaming			
Workload	Local	Global				
Computing	Single-Machine	Distributed				

Background: PowerGraph

Partitioning	Edge-Cut	Vertex-Cut	Hybrid-Cut	Stream-Based	Distributed	Dynamic
Unit	Vertex-Centric	Edge-Centric				
Model	Superstep	Scatter-Gather	Gather-Apply-Scatter (GAS)	Frontier		
Dynamism	Static	Temporal	Streaming			
Workload	Local	Global				
Computing	Single-Machine	Distributed				

Background: Ligra

Partitioning	Edge-Cut	Vertex-Cut	Hybrid-Cut	Stream-Based	Distributed	Dynamic
Unit	Vertex-Centric	Edge-Centric				
Model	Superstep	Scatter-Gather	Gather-Apply-Scatter (GAS)	Frontier		
Dynamism	Static	Temporal	Streaming			
Workload	Local	Global				
Computing	Single-Machine	Distributed				

Background: Problem

- Lack of solutions for single-machine graph processing
- Previous solutions do not take advantage of sequential bandwidth
- Pre-processing (e.g., sorting edge lists) is very expensive
  - Ligra requires pre-processing to produce an inverted edge list
  - Requires random access to a large data structure to switch the edges
    - Source to destination ----> destination to source
  - Dominates overall runtime

Background: X-Stream

Partitioning	Edge-Cut	Vertex-Cut	Hybrid-Cut	Stream-Based	Distributed	Dynamic
Unit	Vertex-Centric	Edge-Centric				
Model	Superstep	Scatter-Gather	Gather-Apply-Scatter (GAS)	Frontier		
Dynamism	Static	Temporal	Streaming			
Workload	Local	Global				
Computing	Single-Machine	Distributed				

Approach: Overview

- Single machine system
- System operates on an abstraction of slow and fast memory
- Data is divided into streaming partitions, which are streamed from slow to fast memory
- Scatter-Shuffle-Gather iterations applied to streaming partitions to implement various algorithms

Approach: Fast and Slow Memory

- Designed for two systems
  1. Graph fits in system memory (**in-memory**)
     - Main Memory -> On-Core Caches
  2. Graph fits in secondary storage (**out-of-core**)
     - SSD/HDD/Magnetic -> Main Memory
- Edge-centric approach identifies that storage offers greater sequential bandwidth than random access bandwidth
  - We stream unordered edges
- Trade-off between a few slower accesses, or many faster accesses

Approach: Streaming Partitions

- A streaming partition consists of:
  - A vertex set
  - An edge list
  - An update list
- The **vertex sets** are mutually disjoint so that their union forms the vertex set of the entire graph
- The **edge lists** contains the edges whose source is in the vertex set
- The **update list** contains all the updates whose destination is in the vertex set
- A fixed number of partitions are chosen based on various constraints

Approach: Edge-Centric Scatter

1. Read vertex set
2. Stream edge list
3. Produce stream of updates

```
scatter phase:
  for each streaming_partition p
    read in vertex set of p
    for each edge e in edge list of p
      edge_scatter(e): append update to Uout
```

Approach: Edge-Centric Shuffle

1. Rearrange updates into update list of streaming partition containing destination vertex

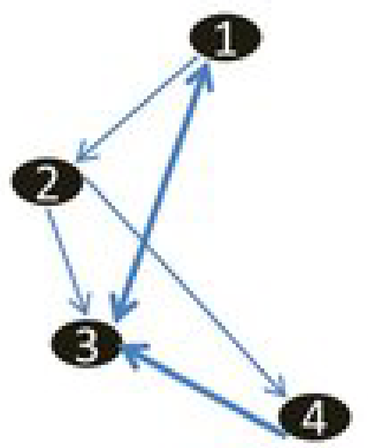
```
shuffle phase:
  for each update u in Uout
    let p = partition containing target of u
    append u to Uin(p)
  destroy Uout
```

Approach: Edge-Centric Gather

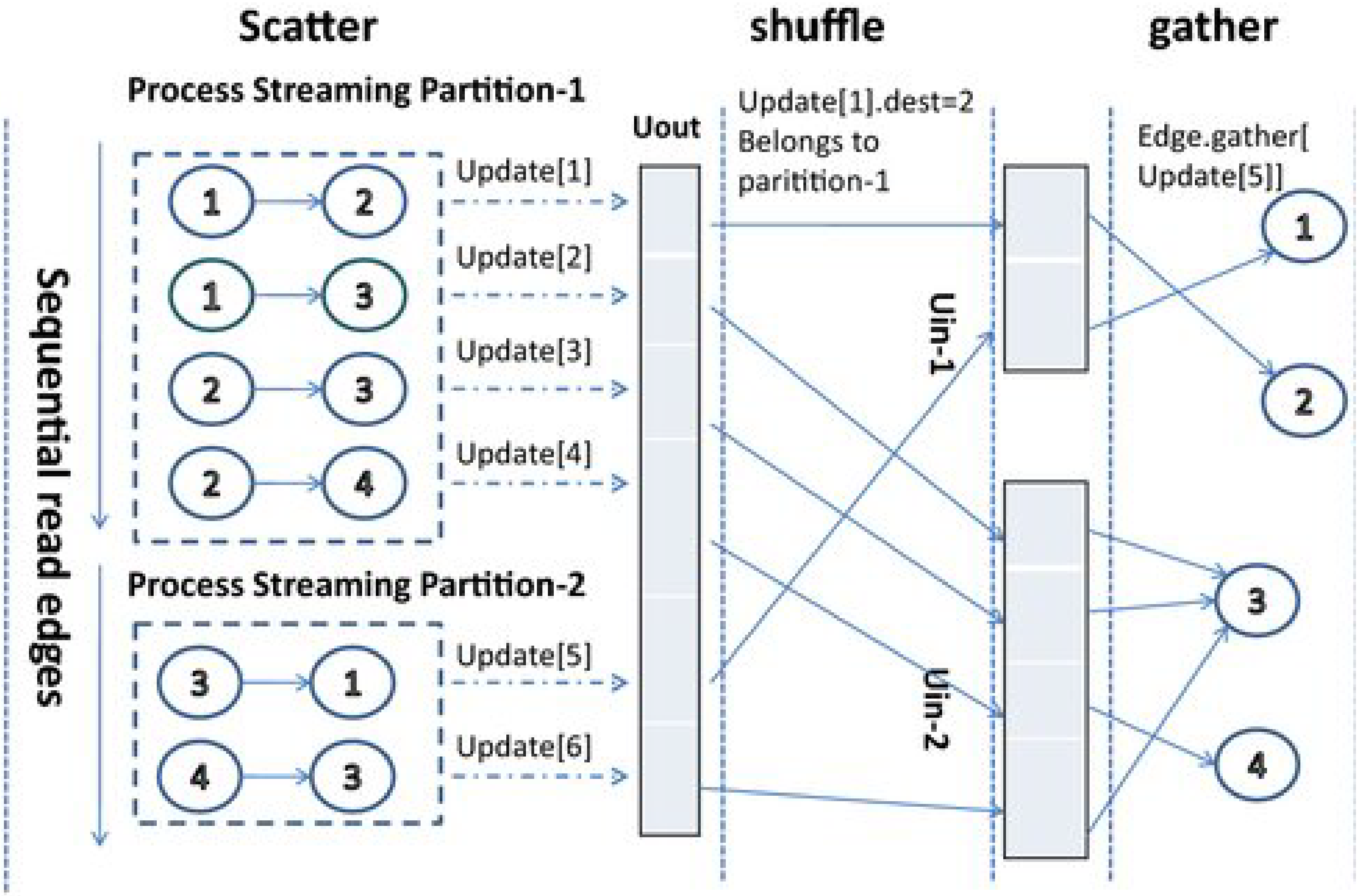
1. Read vertex set
2. Stream update list
3. Compute new vertex values

```
gather phase:
  for each streaming_partition p
    read in vertex set of p
    for each update u in Uin(p)
      edge_gather(u)
    destroy Uin(p)
```

Example



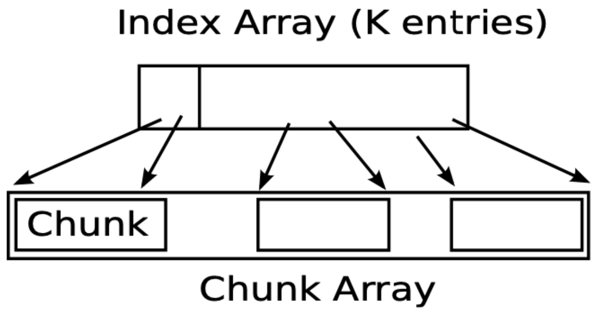
(a) Graph example



(b) Graph data processing

Approach: Out-of-Core Graphs

- We want to achieve sequential access for the shuffle phase as well
- We achieve this by merging the scatter and shuffle phases
- We then use an in-memory buffer to hold updates
  - This is shuffled in-memory when it becomes full
- We also have additional in-memory data structures to hold input from the disk, input/output for the shuffle, and output to the disk

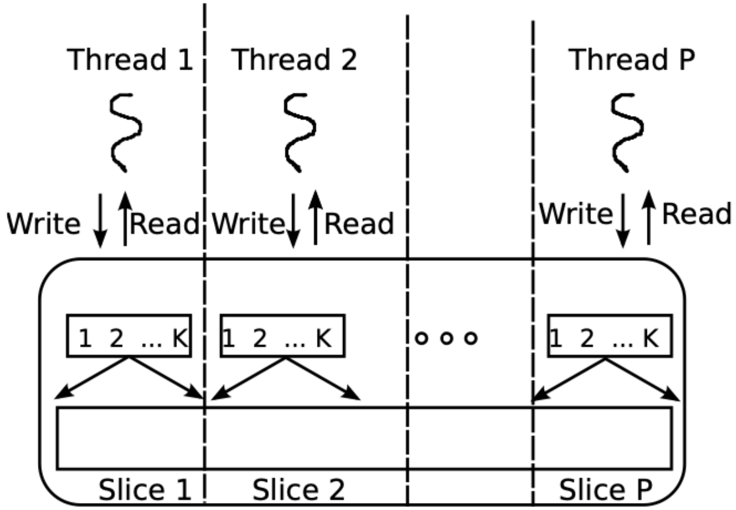



```
merged scatter/shuffle phase:
  for each streaming partition s
    while edges left in s
      load next chunk of edges into input buffer
      for each edge e in memory
        edge_scatter(e) appending to output buffer
        if output buffer is full or no more edges
          in-memory shuffle output buffer
          for each streaming partition p
            append chunk p to update file for p

gather phase:
  for each streaming_partition p
    read in vertex set of p
    while updates left in p
      load next chunk of updates into input buffer
      for each update u in input buffer
        edge_gather(u)
    write vertex set of p
```

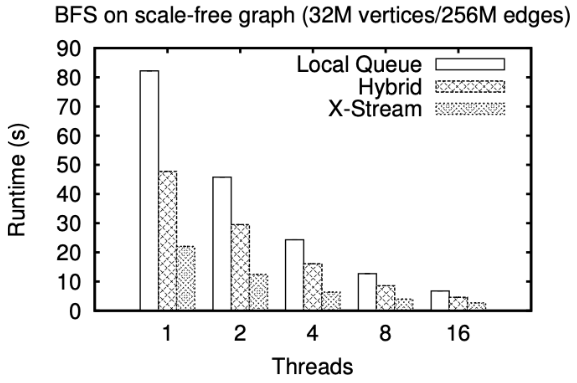
Approach: In-Memory Graphs

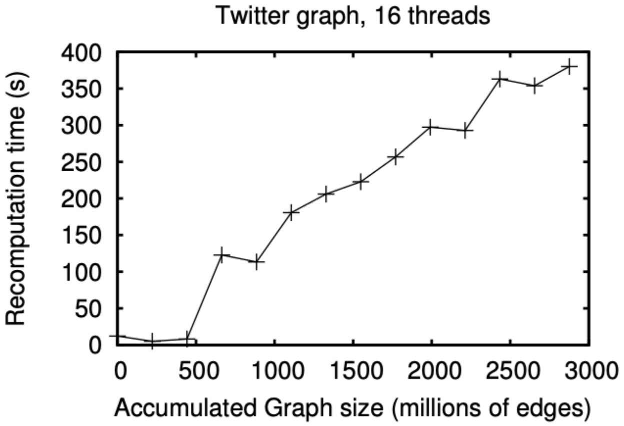
- Utilise all cores to maximise bandwidth
  - In-Memory Streaming Buffer is sliced
  - Parallelisation as threads are executing different streaming partitions – they atomically reserve and fill space in the buffer
- Far more partitions
  - Multi-stage shuffler to deal with increased partition count, as more partitions in a single stage shuffler would lead to challenges in exploiting sequential bandwidth
  - Partitions are placed in a tree hierarchy



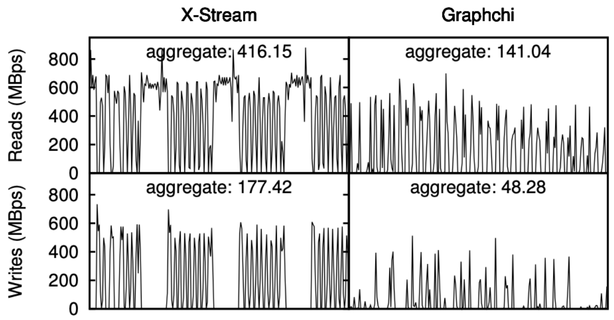
Performance

Threads	Ligra (s)	X-Stream (s)	Ligra-pre (s)
		BFS	
1	11.10	168.50	1250.00
2	5.59	86.97	647.00
4	2.83	45.12	352.00
8	1.48	26.68	209.40
16	0.85	18.48	157.20
		Pagerank	
1	990.20	455.06	1264.00
2	510.60	241.56	654.00
4	269.60	129.72	355.00
8	145.40	83.42	211.40
16	79.24	50.06	160.20





	Pre-Sort (s)	Runtime (s)	Re-sort (s)
Twitter pagerank			
X-Stream (1)	*none*	397.57 ± 1.83	–
Graphchi (32)	752.32 ± 9.07	1175.12 ± 25.62	969.99
Netflix ALS			
X-Stream (1)	*none*	76.74 ± 0.16	–
Graphchi (14)	123.73 ± 4.06	138.68 ± 26.13	45.02
RMAT27 WCC			
X-Stream (1)	*none*	867.59 ± 2.35	–
Graphchi (24)	2149.38 ± 41.35	2823.99 ± 704.99	1727.01
Twitter belief prop.			
X-Stream (1)	*none*	2665.64 ± 6.90	–
Graphchi (17)	742.42 ± 13.50	4589.52 ± 322.28	1717.50



Performance

	WCC	SCC	SSSP	MCST	MIS	Cond.	SpMV	Pagerank	BP
memory									
amazon0601	0.61s	1.12s	0.83s	0.37s	3.31s	0.07s	0.09s	0.25s	1.38s
cit-Patents	2.98s	0.69s	0.29s	2.35s	3.72s	0.19s	0.19s	0.74s	6.32s
soc-livejournal	7.22s	11.12s	9.60s	7.66s	15.54s	0.78s	0.74s	2.90s	1m 21s
dimacs-usa	**6m 12s**	**9m 54s**	**38m 32s**	4.68s	9.60s	0.26s	0.65s	2.58s	12.01s
ssd									
Friendster	38m 38s	1h 8m 12s	1h 57m 52s	19m 13s	1h 16m 29s	2m 3s	3m 41s	15m 31s	52m 24s
sk-2005	44m 3s	1h 56m 58s	2h 13m 5s	19m 30s	3h 21m 18s	2m 14s	1m 59s	8m 9s	56m 29s
Twitter	19m 19s	35m 23s	32m 25s	10m 17s	47m 43s	1m 40s	1m 29s	6m 12s	42m 52s
disk									
Friendster	1h 17m 18s	2h 29m 39s	3h 53m 44s	43m 19s	2h 39m 16s	4m 25s	7m 42s	32m 16s	1h 57m 36s
sk-2005	1h 30m 3s	4h 40m 49s	4h 41m 26s	39m 12s	7h 1m 21s	4m 45s	4m 12s	17m 22s	2h 24m 28s
Twitter	39m 47s	1h 39m 9s	1h 10m 12s	29m 8s	1h 42m 14s	3m 38s	3m 13s	13m 21s	2h 8m 13s
yahoo-web	—	—	—	—	—	16m 32s	14m 40s	1h 21m 14s	8h 2m 58s

(a)

	# iters	ratio	wasted %
memory			
amazon0601	19	2.58	63
cit-Patents	21	2.20	50
soc-livejournal	13	2.13	57
dimacs-usa	**6263**	1.94	**98**
ssd			
Friendster	24	1.06	63
sk-2005	25	1.04	67
Twitter	16	1.04	55
disk			
Friendster	24	1.04	63
sk-2005	25	1.04	67
Twitter	16	1.04	55
yahoo-web	—	—	—

(b)

Graph	**# steps**
In-memory	
amazon0601	19
cit-Patents	20
soc-livejournal	15
dimacs-usa	**8122**
Out-of-core	
sk-2005	28
yahoo-web	**over 155**

Evaluation: Positives

- Single-machine lowers barrier to entry
  - Compared to large-scale distributed systems, at the cost of size capacity
- Exploits sequential bandwidth
- Flexible framework for in-memory and cache
- Performant for certain applications
- Removes need for sorting
  - Major consideration for other systems

Evaluation: Negatives

- The Scatter-Gather programming model can be restrictive
  - X-Stream supports more, but the paper does not go into detail
- Parallelism for in-memory graphs is dependent on effective work distribution
  - X-Stream supports work stealing, but the paper does not go into detail
- Some issues in evaluation
  - Comparisons against other systems is very limited and seems cherry-picked
  - Especially inadequate in comparing to well-established distributed systems, which would ground results
- There is no discussion of fault tolerance, which suggests it was not a major consideration
  - Makes the system unsuitable for many applications
- Does choosing sequential over random access have power consumption implications?

Thank You