X-STREAM:

Edge-centric Graph Processing using Streaming Partitions

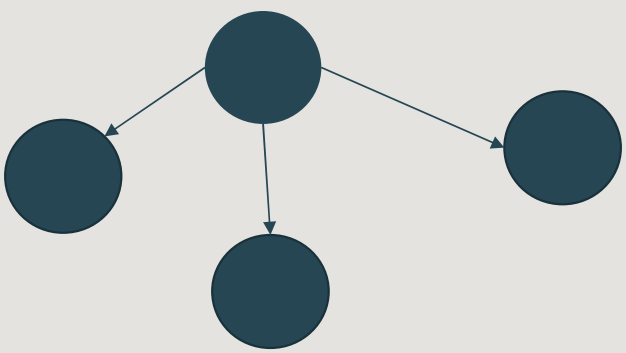
Authors: Amitabha Roy, Ivo Mihailovic, Willy Zwaenepoel

Why Previous Solutions Do Not Scale

Random Access

1	2	3	4	5	6	7	8	9	…

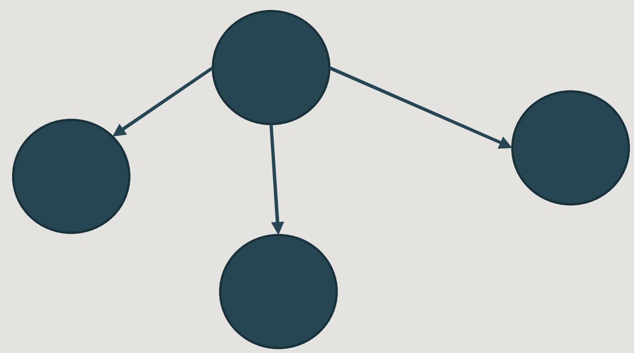
Vertex-Centric



Sequential Access

1	2	3	4	5	6	7	8	9	…

Edge-Centric



Vertex-Centrism

```
Scatter(v : vertex):
    Send(Outgoing[V])
```

```
Gather(v : vertex):
    Apply( Incoming[V])
```

Requires edge and vertex data in fast memory

Allows for pre-processing/sorting edge data for faster algorithms

Problems

- **Most Graphs have significantly more edges than vertices**

- **Harder to partition graph data**
- **Random access over both vertices and edges**

- **Pre-processing dominates the run-time**

Edge-Centrism

Scatter(e: edge):
    Send_Update(e)

Gather(u :update):
    Apply(u, u.dest)

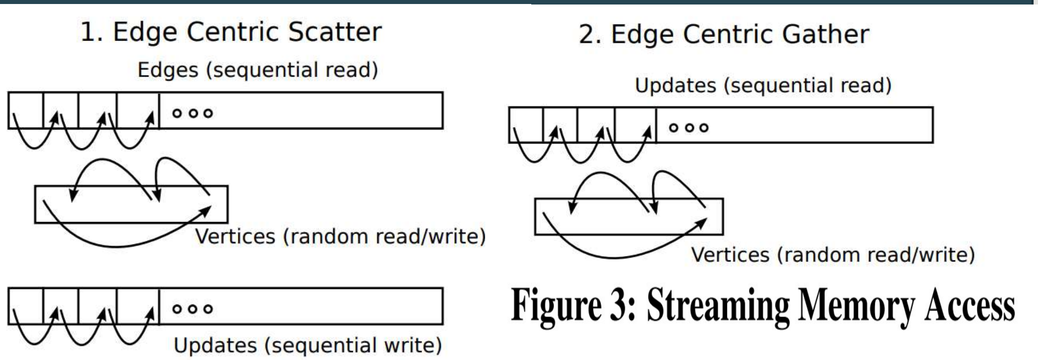


Figure 3: Streaming Memory Access

Requires almost zero pre-processing

Only needs fast random access to vertex data

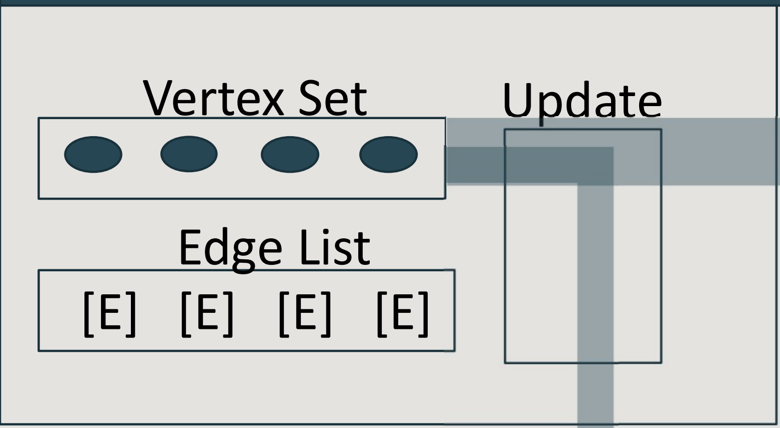
Benefits

- Better mapping to hardware and the structure of real graphs

- Allows for streaming the edge data from slow memory sequentially, speed-up:
  - Disk: 500x
  - SSD: 300x
  - RAM: 4.6x/1.8x for 1/16 cores

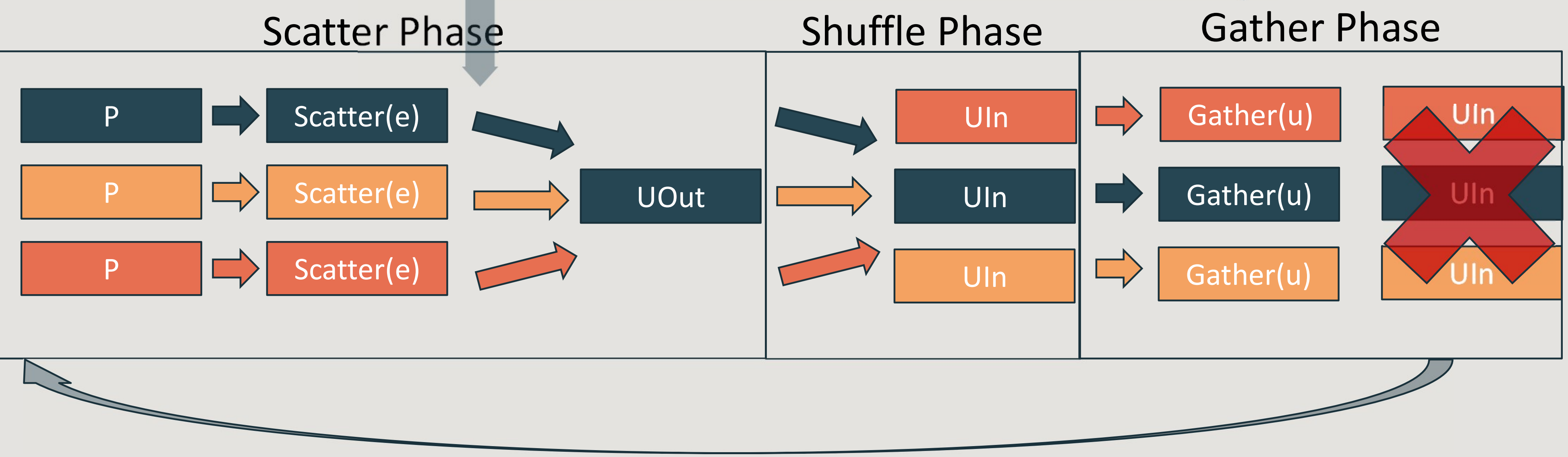
- Better initial run-time performance
- No bottlenecks in maintaining invariants

Streaming Partitions



Basic mapping to hardware:

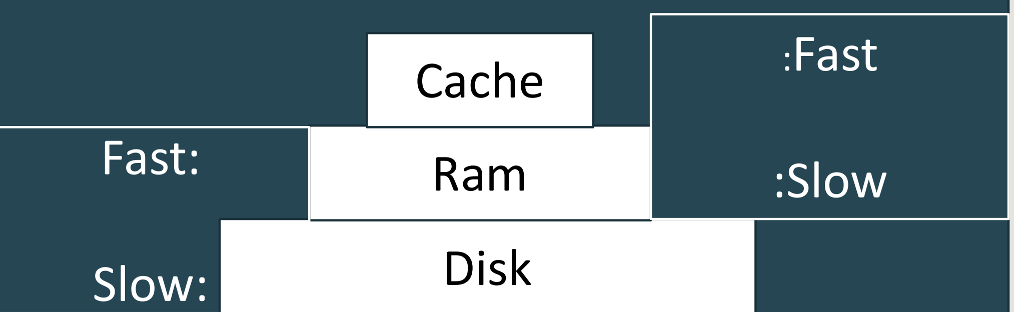
- **Vertex set should fit into fast storage**
- **Assume uniform distribution of updates**
- **Divide the graph uniformly**
  - **Considering auxiliary data structures**
  - **Algorithms are order-independent**



Hierarchical Memory

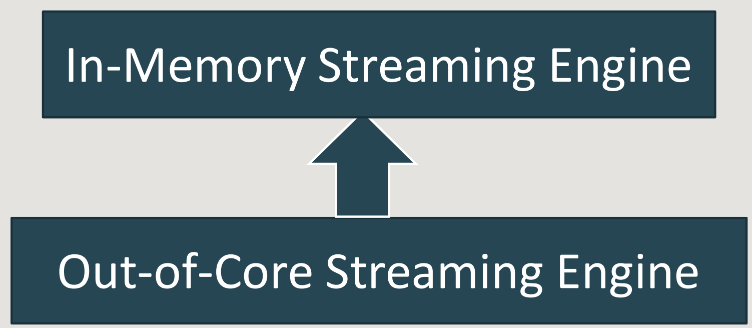
Two types of relative storage: Slow vs Fast

Partitioning the Memory Hierarchy:



Processing

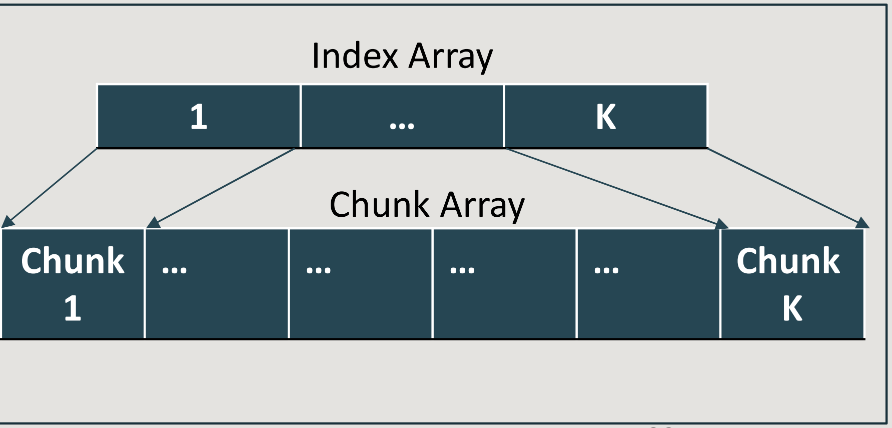
- **X-Stream implements streaming engines for handling transfer from slow to fast storage**
  - **Making heavy use of large static streaming buffers to carry data**



Out-of-Core Streaming Engine

Moves data from disk to memory

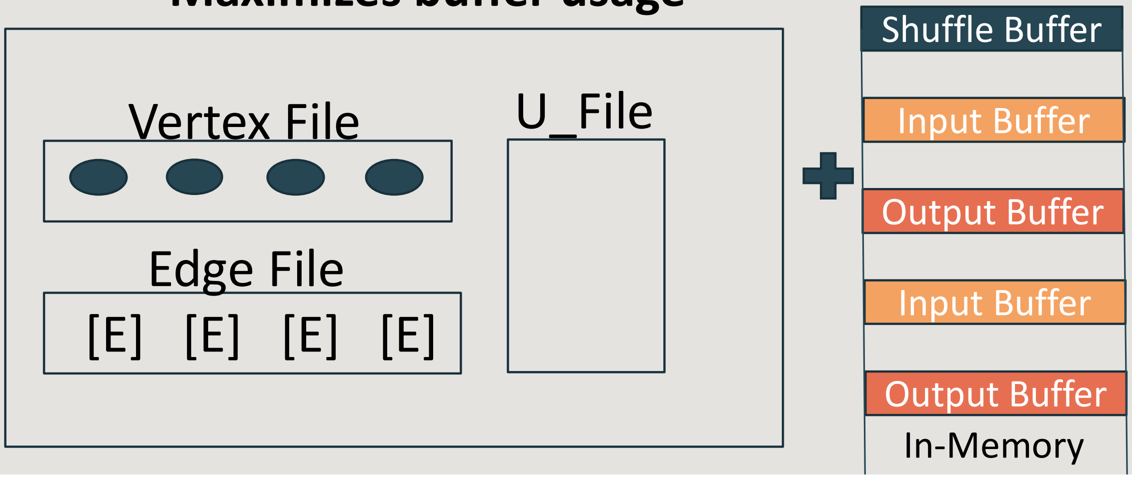
- **In order to transfer data from disk to memory it uses a simple stream buffer**
- **Partition size and buffer are both statically allocated**



K-Partition Stream Buffer

Modified computation model

- **All incoming/outgoing data from/to disk passes through in/out buffers(2 of each for prefetching)**
- **Shuffle stages are performed within scatter phases whenever UOut becomes full**
  - **Maximizes buffer usage**



In-Memory Streaming Engine

Must be able to do parallel computation on streaming buffers:

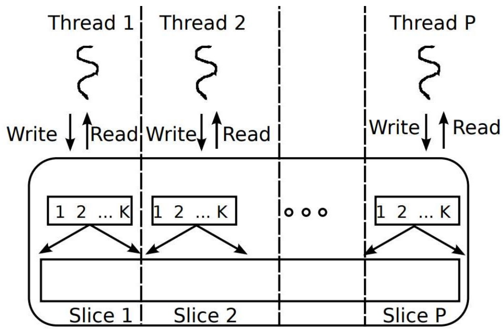


Figure 7: Slicing a Streaming Buffer

Also parallelizes the scatter -> shuffle -> gather pipeline along stream buffers

- Required implementing work-stealing as streaming partitions differ in edge counts

Implementation Details

Parallel Multistage Shuffler:

- **Arranges partitions into a tree structure**
- **Uses a power of two for both the number of partitions and fanout**
- **Inputs get shuffled by being passed down the tree and split up at every step**

Layered Approach:

- **Sits above the disk streaming layer**
- **Disk layer operates as normal, however the in-memory processing of a partition is further fed into the in-memory system**

Performance Evaluation Results

Wasted Computation:

	# 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)

This is a direct result of a large-diameter structure

Wasted computation is an expected trade-off from large-scale streaming

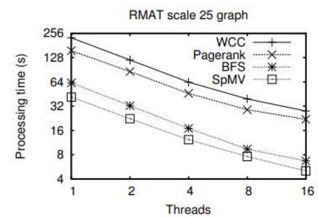


Figure 14: Strong Scaling

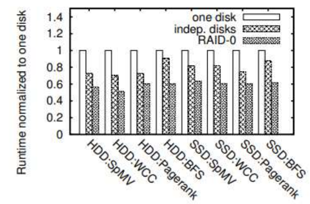


Figure 15: I/O Parallelism

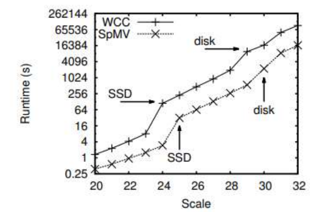


Figure 16: Scaling Across Devices

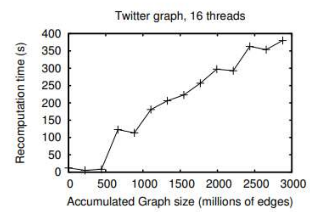


Figure 17: Re-computing WCC

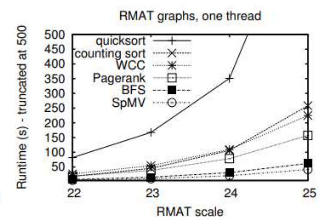


Figure 18: Sorting vs. Streaming

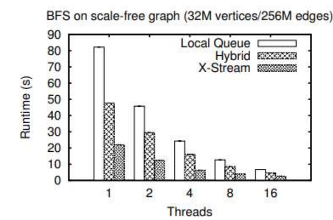


Figure 19: In-memory BFS

Sorting dominates the run-time of most systems we will see

System scales very well on most tasks, linearly until a new storage medium is needed

Ligra And Graphchi Comparison

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

Figure 20: Ligra [48] on Twitter (99% CI under 5%)

Pre-processing in Ligra takes longer than the entire X-Stream execution

	BFS [33]	X-Stream
IPC	0.47	1.30
Mem refs.	982 million	620 million
	Ligra,BFS [48]	X-Stream
IPC	0.75	1.39
Mem refs.	1.3 billion	1.5 billion

Figure 21: Instructions per Cycle and Total Number of Memory References for BFS

The efficiency of sequential memory access also makes X-Stream dominate in IPC

Overall, Ligra should still massively overperform on speed in most real use-cases

	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

Figure 22: Comparison with Graphchi on SSD with 99% Confidence Intervals. Numbers in brackets indicate X-Stream streaming partitions/Graphchi shards (Note: re-sorting is included in Graphchi runtime.)

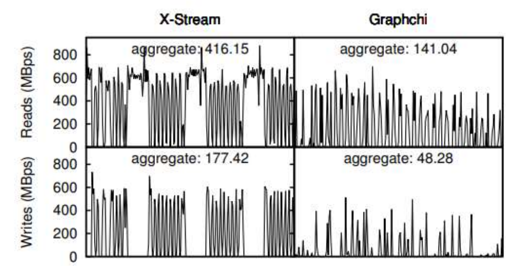


Figure 23: Disk Bandwidth

Graphchi serves a similar use case to X-Stream while applying a vertex-centric model. The average speed-up without pre-processing is 2.3 and 3.7 with pre-processing.

Disk bandwidth usage is also more predictable in X-Stream.

Opinion/Motivation

Can Sorting Keep-Up?

- Any vertex-centric computation requires some way of associating edges to source/destination vertices, sorting is the most popular
- Sometimes it is necessary to look at a reversed edge-list for classes of algorithms
  - Requires either re-sorting repeatedly or maintaining two views of the edge list.
- This narrative *has been extensively challenged* by Frank McSherry using radix sort to process twitter data 10x faster than the X-Stream authors estimation

- Vertex-Centric: Edge Data/RAM Bandwidth
- Edge-Centric: Scatter X E_Data/Seq Band

Real-World Graphs:

- All of the scale-free graphs perform very well with X-Stream, many real-world graphs follow a power-law distribution.
- Work stealing seems sufficient to handle high-degree vertices.
- Real world graphs grow very slowly in diameter O(log(V)/log(log(V)) and can even undergo densification

Context

Creation:

- According to Amitabha Roy in “X-Stream: A Case Study in Building a Graph Processing System”
- The algorithms used within the system were first devised by observing the relation between graph processing and sparse matrix-vector multiplication
  - Followed by applying advances in SpMV to graph processing
- The implementation, systems and evaluation were subsequently developed for publishing in “Symposium on Operating Systems Principles”
  - The final paper changes the focus to the systems aspect of X-Stream

Development History:

- The GitHub has not had any commits in years
- Authors from EPFL also developed “Chaos” as the multi-machine successor of X-Stream, utilizing many of the same ideas surrounding streaming partitions with a heavy focus on work stealing and ignoring locality
  - The new system is capable of handling graphs with 1 trillion edges ~ 16 TB of data
  - Later scaled to 8 trillion on only 32 machines
- “Chaos” development, at least publicly, also seems to have ceased soon after creation

Why Has It Not Had a Larger Impact?

- Unusual edge-centric computational model
- Algorithmic origins:
  - Tumultuous implementation
  - Potentially difficult to extend or maintain
- No way of easily changing graph structure, relies on static data structures
- No long-term support
- Lacks comprehensive documentation, high-level means of integration, or a killer-app
- Highly focused towards throughput and cost over speed, niche use-case
  - As shown by Frank McSherry, for some specific tasks, better algorithms implemented with less-restrictive programming models and efficient pre-processing may be superior.
- In my opinion, it was never intended for production
- As an academic work it has a fair number of citations and inspired systems
- Similar critiques apply to “Chaos”

Questions?