R244: Mini-Project

Evaluating **X-Stream**: Edge-centric Graph Processing using Streaming Partitions

Relevant Paper: Roy et al., EPFL, 2018

Jan Pytel
University of Cambridge
December 3, 2025

X-Stream

- Graph processing system for:
  - in-memory graphs
  - out-of-core graphs
- Single, shared-memory machine
- Scatter-Gather programming model
- Novelty
  - **edge-centric** (as opposed to vertex-centric)
  - **streaming unordered edge list**
    - from RAM (in-memory graphs)
    - from disk (out-of-core graphs)
- Motivation: **sequential access** > random access bandwidth

Scatter-Gather

- Given a graph (vertices with state connected by edges)
- Iterative computation:
    1. scatter vertex state to neighbours
    2. gather updates from neighbours and recompute vertex state
- Vertex-Centric implementation
    - iterates over vertices
    - issue: random access for edges
- Edge-Centric implementation (**X-Stream**)
    - iterates over edges
    - benefit: sequential access to edge list
- Example algorithms: BFS, Shortest-Paths, PageRank

Slow Storage and *Fast Storage*

	In-Memory Graphs	**Out-of-Core Graphs**
Fast Storage	Cache	RAM
Slow Storage	RAM	SSD/disk

Streaming Partitions

- Idea: split vertex set into subsets that fit into *Fast Storage*
- Consists of
    - **vertex set** – subset of vertices
    - **edge list** – edges whose source vertex is in the partition's vertex set
    - **update list** – updates whose destination vertex is in partition's vertex set
        - recomputed before each gather phase
- Size of Streaming Partition ~= size of *Fast Storage*
    - (allowing for buffers and additional data structures)
- Processing
    - streams the partitions from *Slow* into *Fast Storage*
    - computes the scatter-gather on partitions in *Fast Storage*

X-Stream Results

- Outperforms existing graph processing systems
- Key performance factors
  - sequential access
  - no pre-processing cost (e.g. sorting & indexing)
  - higher count of instruction per cycle (lower memory resolution latency)
- Scalability, good across:
  - number of cores
  - number of I/O devices
  - different storage devices

My Mini-Project Motivation

- Even though X-Stream itself has not been adopted
- Philosophy: **sequential I/O** and **edge-centric approach** has motivated subsequent research
  - I would like to investigate and evaluate that further
- My MPhil focus: systems & networking
- Interest in how hardware characteristics shape system design

Progress so Far

- X-Stream repository: https://github.com/bindscha/x-stream
  - engine implemented in C++
  - quite old, 2015
  - cloned and built it up on MacOS laptop
- Repository includes
  - RMAT & random synthetic graph generator
  - Several pre-implemented algorithms
    - PageRank, BFS, CC, SSSP, SpMV, MIS, triangles, etc.
- Generated example RMAT graph
  - directed, 1M vertices, 16M edges
- Executed PageRank on this graph
  - (10 iterations, 16 cores, 256MB memory)
- Collected stats like: #edges streamed, #updates, I/O bytes read & written, RSS usage

Planned Future Work

- Move to Linux for accurate evaluation
  - MacOS does not support Linux's O_DIRECT (crucial)
- Use real graphs (not only synthetic)
- Evaluate multiple algorithms
- Reproduce key experiments from X-Stream paper, e.g.
  - sequential streaming throughput
  - memory bandwidth with threads
  - disk bandwidth with buffer size
  - scaling with number of threads
  - performance on long-diameter graphs (X-Stream's limitation)

Planned Future Work

- Additional experiments
  - Direct I/O vs Pagecache (--no-dio mode)
    - X-Stream is designed to work best with graphs of size much larger than memory
    - for smaller graphs Pagecache mode might actually perform better
    - experiment: vary graph size and compare direct I/O vs Pagecache
    - goal: investigate where the boundary lies
  - Autotuner vs Forced In-Memory (--force_buffers 2)
    - autotuner (chooses streaming partition sizes) assumes graphs are disk-bound and optimises partitioning for streaming from disk
    - --force_buffers 2 can force the workload into memory and avoid disk I/O
    - experiment: vary graph size and compare autotuner vs --force_buffers 2
    - goal: find when in-memory mode outperforms disk-optimised mode
      - and how this depends on graph size and algorithms
  - …