Graph Processing System with Multi-level Architecture

MIKHAIL CHERNOSKUTOV

IMM UB RAS, URFU

E-MAIL: MACH@IMM.URAN.RU

Modern systems for (big) graph processing

Parallel Boost Graph Library, Pregel, CuSha, GraphCT, NetworkX, PowerGraph, graph-tool, GraphBLAS, KDT, igraph, STINGER, Ligra, Gunrock, HelP, GPS, Galois, Green-Marl, Gephi, Medusa, MapGraph, NetworKit, SNAP, GraphLab, Giraph, JUNG, Pajek, GraphPad, PEGASUS, GraphX, GraphChi, Totem, Vertexapi2

+ a lot of papers dedicated to performance engineering of well-known graph algorithms

Classification of graph processing systems

Different graph processing models

- Vertex-centric
- Domain-specific languages
- Processing primitives

Vertex-centric model

"Thinking like a vertex"

Each vertex
- Has some data about itself, ingoing and outgoing edges and make some computations
- Use ingoing edges to receive messages from other vertices
- Use outgoing edges to send messages to other vertices

Pros
- Natural way to parallelize your application

Cons
- Bad suitable for some algorithms (adjacency matrix-based)

First implementation
- Pregel (Google, 2010)

Doman-specific Language

Domain-specific language is a computer language specialized to a particular application domain.

- User develops program using specific domain terminology
- Compiler translates DSL code to target programming language (for instance, C++ or CUDA)

Pros

- Increasing developer productivity
- Cross-platform

Cons

- It is hard to integrate DSL code in application that developed using other programming language

Doman-specific Language

Green-Marl – DSL for graph processing on shared memory systems

- Has C/C++ compiler

Other implementations

- PowerGraph
- Galois
- GraphChi
- GraphLab

```
Procedure Compute_BC(
 G: Graph, BC: Node_Prop<Float>(G)) {
  G.BC = 0;        // initialize BC
  Foreach(s: G.Nodes) {
   // define temporary properties
    Node_Prop<Float>(G) Sigma;
    Node_Prop<Float>(G) Delta;
    s.Sigma = 1; // Initialize Sigma for root
   // Traverse graph in BFS-order from s
    InBFS(v: G.Nodes From s)(v!=s) {
      // sum over BFS-parents
      v.Sigma = Sum(w: v.UpNbrs) {w.Sigma};
    }
   // Traverse graph in reverse BFS-order
    InRBFS(v!=s) {
      // sum over BFS-children
      v.Delta = Sum (w:v.DownNbrs) {
         v.Sigma / w.Sigma * (1+ w.Delta)
      };
      v.BC += v.Delta @s; //accumulate BC
} } }
```

Parallel processing primitives

Basic idea

- Select common graph operations
- Implement it as parallel highly optimized building blocks
- Develop graph algorithms as combinations of such primitives

Pros

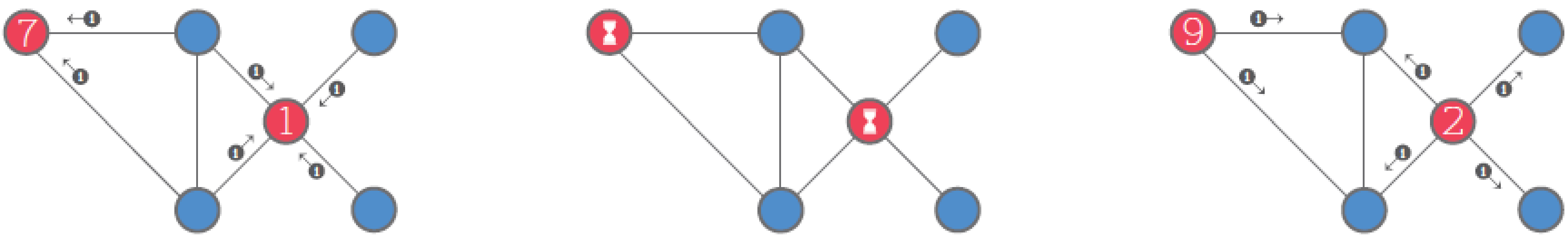
- Simplification of development and debugging
- Developed on common programming languages

Cons

- There is no single complete set of primitives

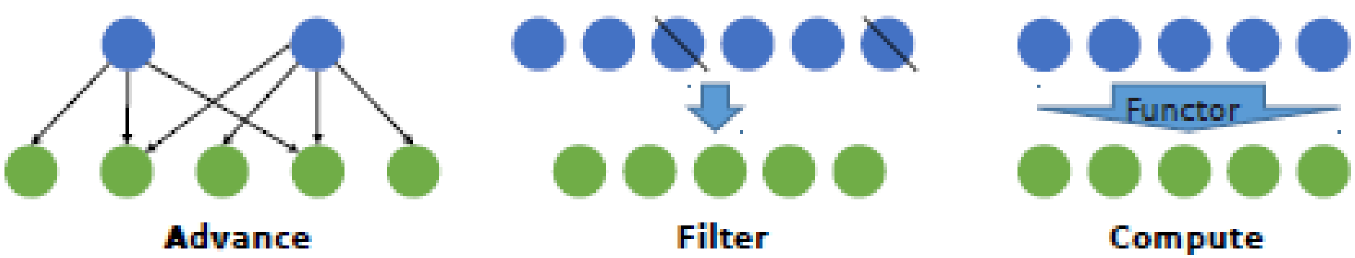
Parallel processing primitives

Gather-Apply-Scatter (MapGraph, PowerGraph)



Gather Apply Scatter

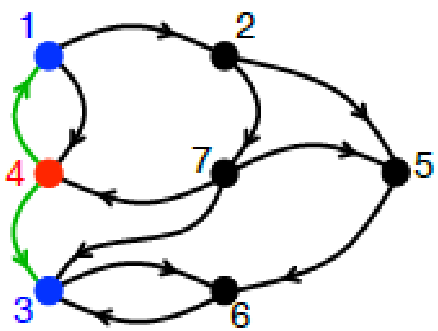
Advance-Filter-Compute (Gunrock)

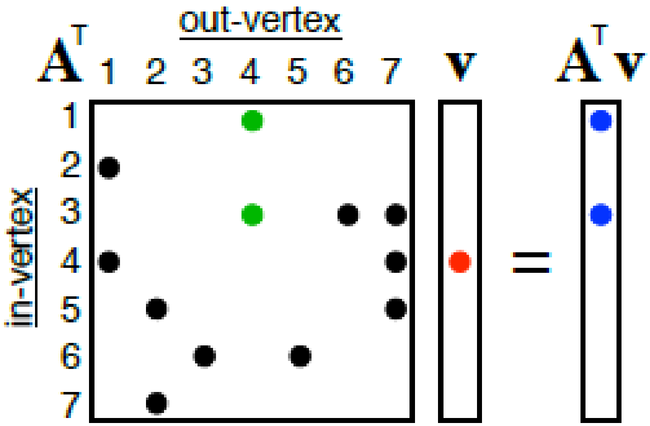


Parallel processing primitives

GraphBLAS

- Attempt to describe graph algorithms on the language of linear algebra
- Under development since 2008 year





Real-world problems meet graph processing software

Most popular systems are sequential and based on languages like Python, R, etc.

Why researchers don't use parallel big graph processing systems?

- Choose only one feature from the list…
  - Support of various architectures
  - High performance processing
  - A lot of implemented algorithms

My network science

Natural language processing / financial transactions processing

- Detect sets of similar things in network
  - ~100 000 vertices and edges
- Overlapping community detection
  - Find and rank k-cliques

City logistics

- Graphs with parallel edges
  - ~ 100 000 vertices and millions of edges
- Algorithms like max-flow which are sensitive to graph data structure
- Dynamically changing graphs

“Perfect” graph processing system

Ability to use different graph data structures to tune application performance

- Fast topology modification
- Perfect data structure can do O(1)

A lot of implemented algorithms

Build graph processing algorithm using functions

Ability of parallelization

Proposed multi-layer architecture

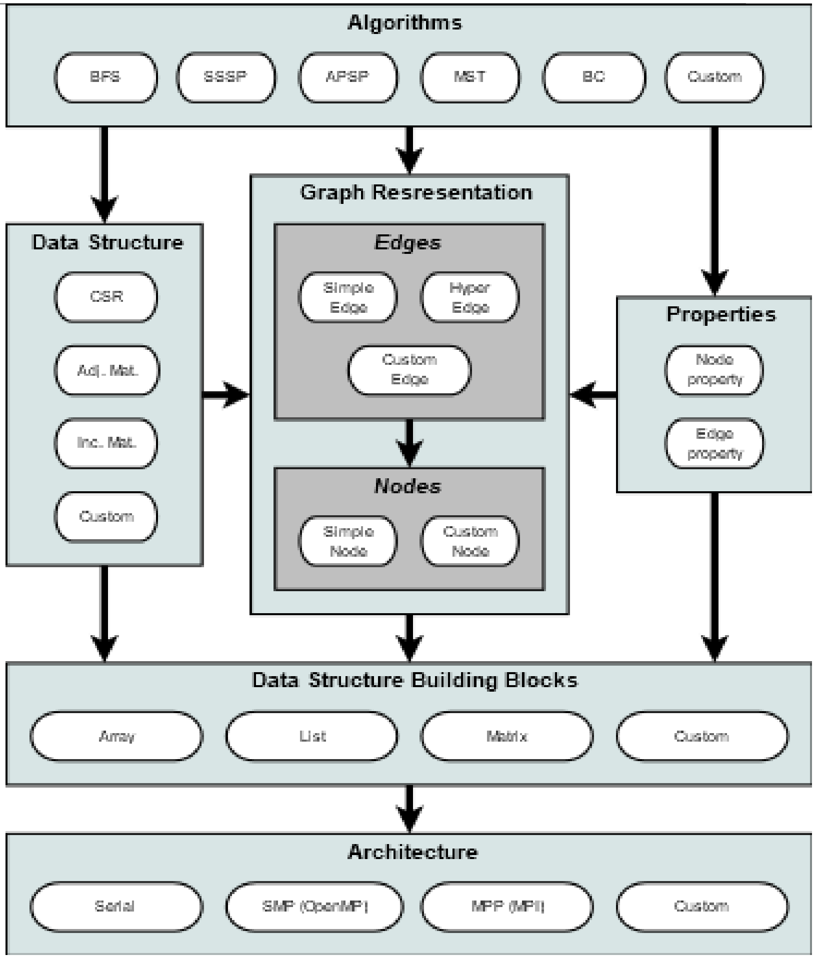
Algorithms level
- High-level operations on graph

Graph representation
- Storage for nodes and edges

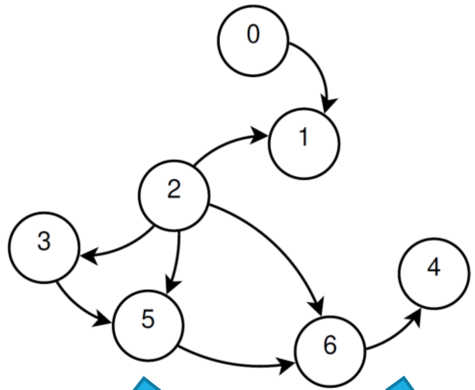
Data structure
- Organize nodes and edges for efficient read and write operations

Properties
- Any edge/node property

Data structure building blocks
- “Atomic” data structures that used for construction of graph data structure



Benchmarking (1)



$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Compressed Sparse Rows (CSR):

- row pointers = [0, 1, 1, 5, 6, 6, 7, 8]
- column ids = [1, 1, 3, 5, 6, 5, 6, 4]

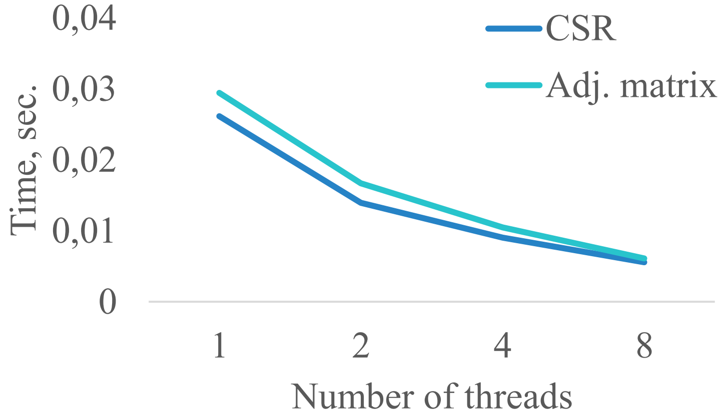
Benchmarking (2)

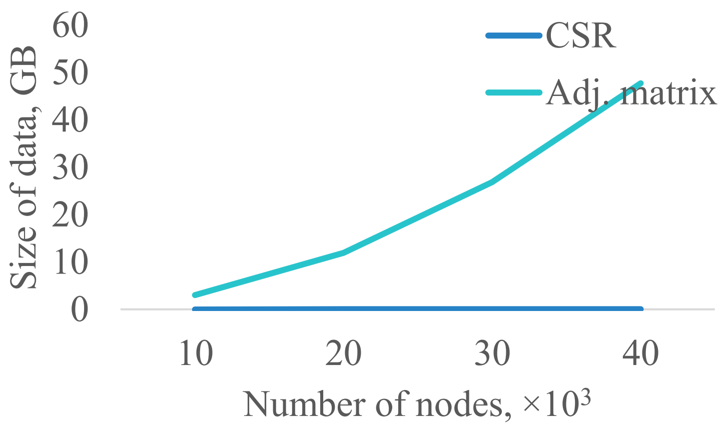
C++ implementation

BFS algorithm with CSR and Matrix data structures

RMAT graph ~50×10^3 nodes and 16 edges per node

OpenMP scalability up to 8 threads

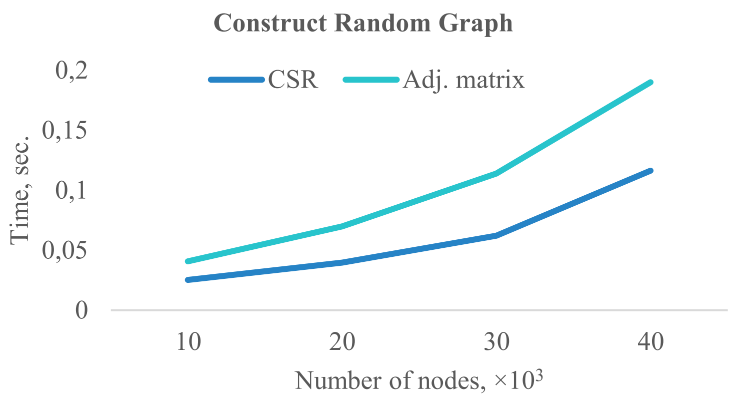


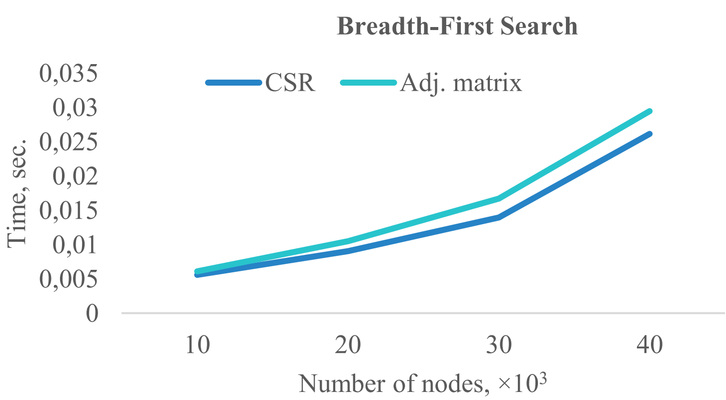


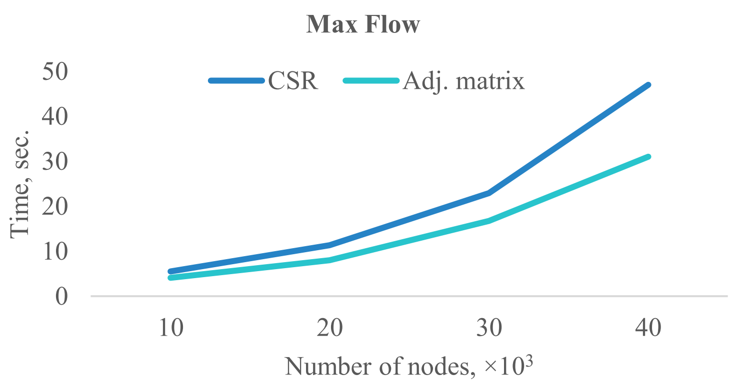
Benchmarking (3)

Different algorithms with CSR and Matrix data structures

RMAT graphs from ~10×10³ to ~40×10³ nodes







Future research

More efficient data structures

- Navigation and modification with linear complexity

More algorithms

- Paths, flows, centralities, communities, etc.

MPI parallelization

- Adopt graph processing system for MPP parallelization

Parallelization

Development complexity

	SMP	**MPP (less than 50 nodes)**	**MPP (more than 50 nodes)**
Easy	NetworkX, igraph	ND	ND
Medium	Now	To be done	GraphBLAS
Hard	C/C++ development	ND	Parallel BGL

Questions?