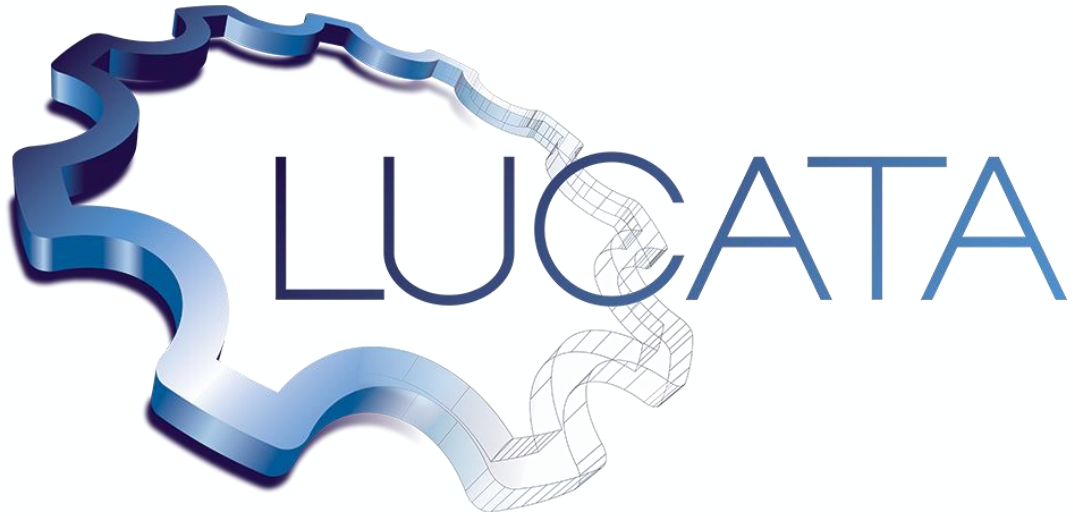
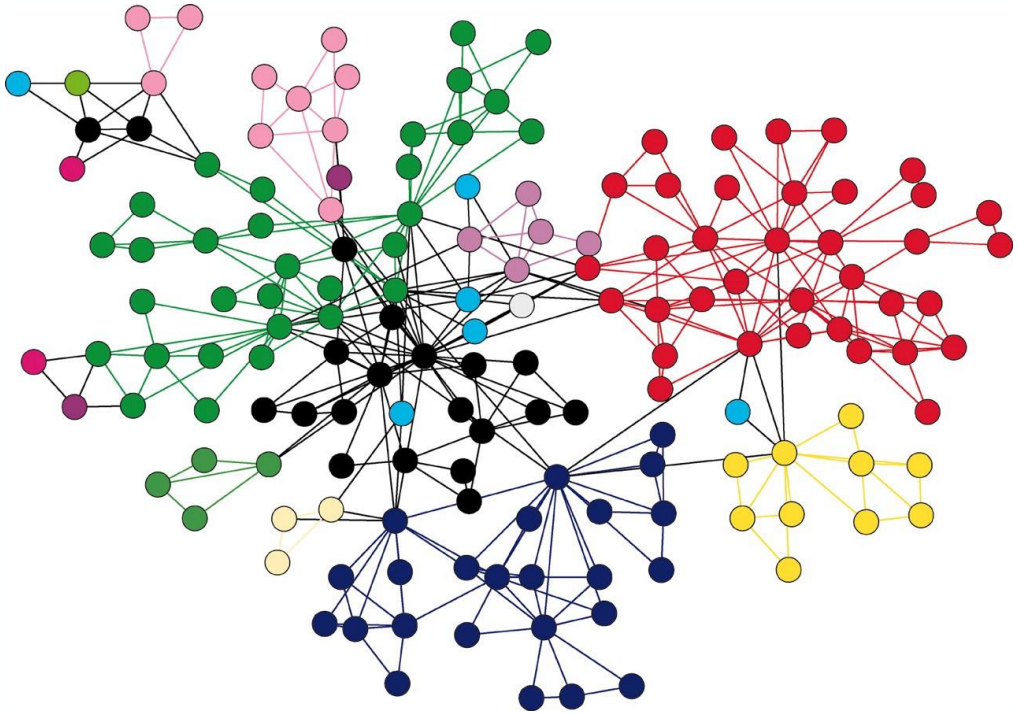


Graph Algorithms and GraphBLAS

Graph Data Structure

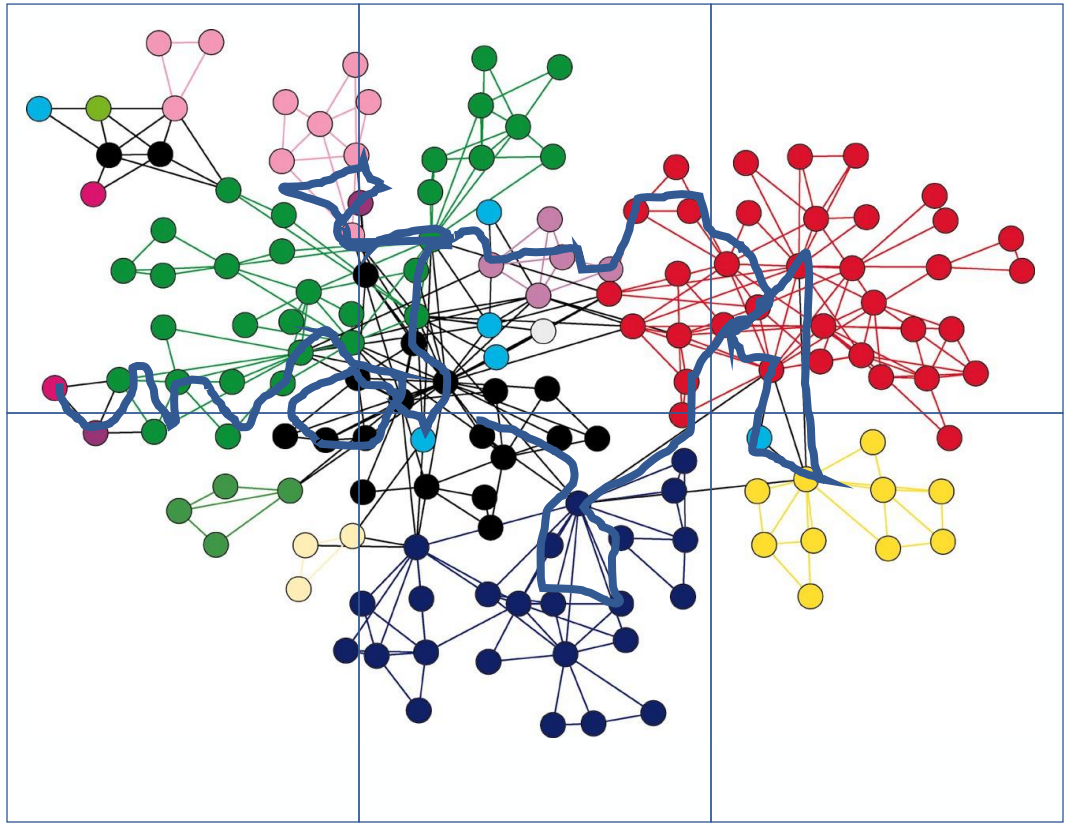


Set of N vertices
V = {0,1,2,…,N-1}
Set of edges between them
E = {(0,1),(3,9),(1,2),…}
Each vertex has information about a certain entity
Each edge denotes a relationship between entities

Extracting information from the graph involves accessing numbers in the pattern of the graph (e.g., *traversing* the graph)

How do we partition and store the graph in a manner that minimizes the impact of memory access on algorithm performance?

Block Graph Partitioning

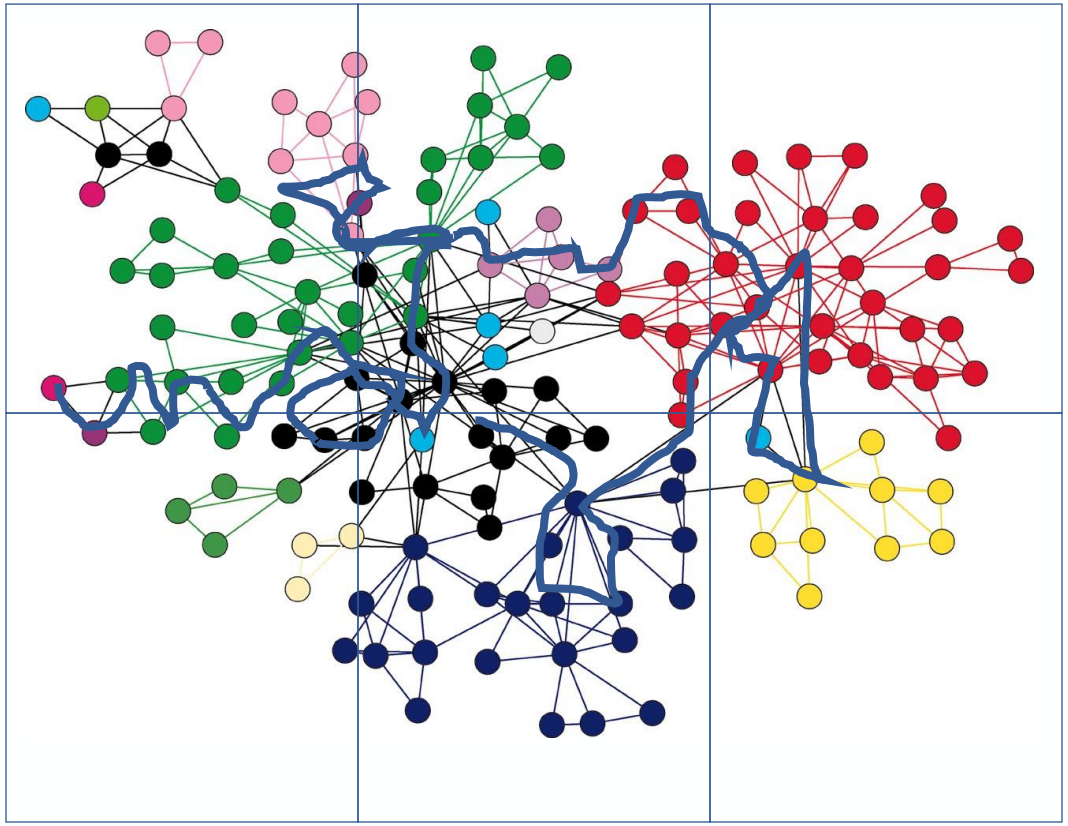


Vertices in the graph are arranged into blocks of memory
Edges from a vertex are stored in the same block as that vertex

Traversing the graph will involve small accesses to multiple blocks of memory in non-regular sequences

Extremely inefficient for caches (little reuse within blocks) and distributed memories (lots of network traffic)

Lucata Performance Advantage



Migrating threads reduce network traffic by avoiding a round-trip memory access

Narrow channel memory improves memory efficiency by avoiding unnecessary data movement

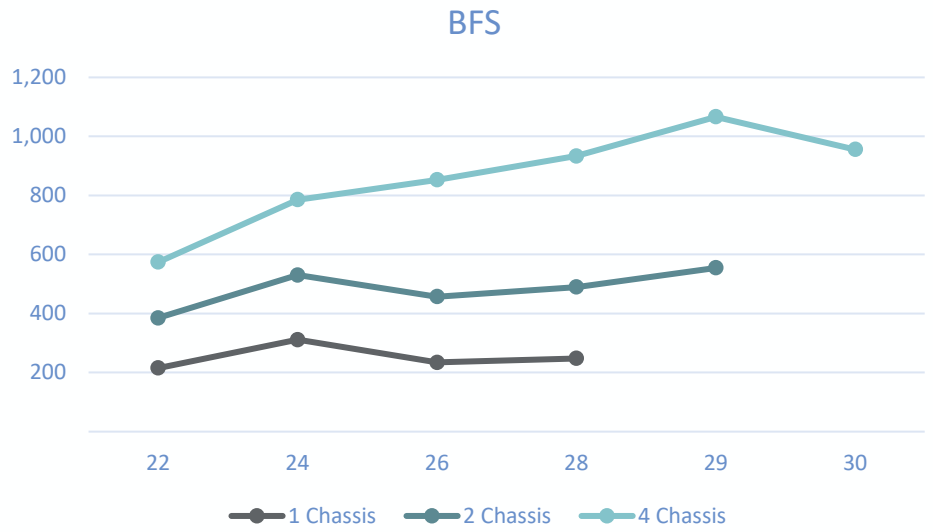
Lucata offers performance advantages for graph algorithms that far outstrip what can be achieved by traditional architectures

Application Areas and Graph Size

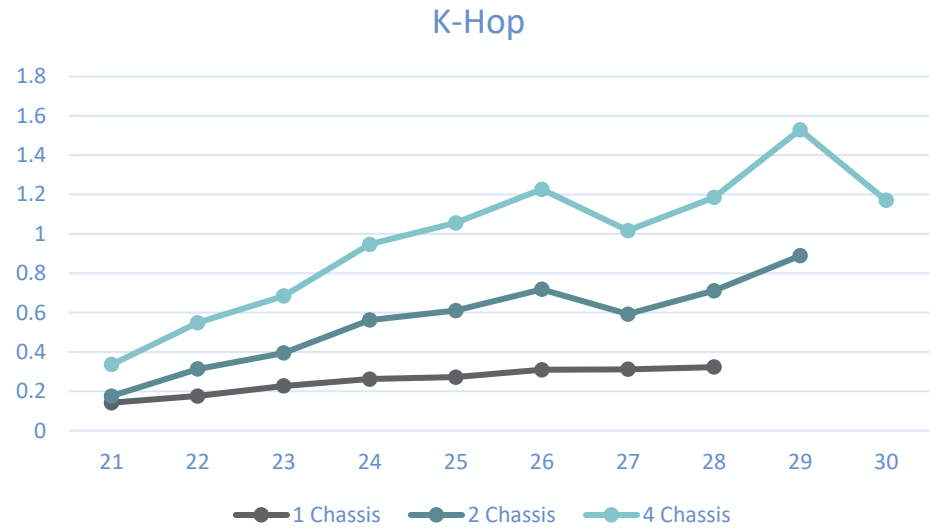
- **Anomaly Detection**
  - Credit card fraud
  - Money laundering
  - Malware detection
- **Community Identification**
  - Recommendation engines
  - Identification of bad actors (terrorist)
  - Insider trading
- **Trend Detection**
  - Security price movements

Best uses for Lucata technology are applications with very large (i.e., many nodes) and sparse (i.e., many nodes, fewer edges) graphs with real-time constraints

Lucata Pathfinder - Performance on Graph Algorithms



*** Data in MTEPS (millions of edges traversed per second)**



*** Data in GTEPS (billions of edges traversed per second)**

Highly scalable platform for graph algorithms

Graph Software Packages

LUCATA

Standards for Graph Primitives

Tim Mattson (Intel Corporation), David Bader (Georgia Institute of Technology), Jon Berry (Sandia National Laboratory), Aydin Buluc (Lawrence Berkeley National Laboratory), Jack Dongarra (University of Tennessee), Christos Faloutsos (Carnegie Melon University), John Feo (Pacific Northwest National Laboratory), John Gilbert (University of California at Santa Barbara), Joseph Gonzalez (University of California at Berkeley), Bruce Hendrickson (Sandia National Laboratory), Jeremy Kepner (Massachusetts Institute of Technology), Charles Leiserson (Massachusetts Institute of Technology), Andrew Lumsdaine (Indiana University), David Padua (University of Illinois at Urbana-Champaign), Stephen Poole (Oak Ridge National Laboratory), Steve Reinhardt (Cray Corporation), Mike Stonebraker (Massachusetts Institute of Technology), Steve Wallach (Convey Corporation), Andrew Yoo (Lawrence Livermore National Laboratory)

Abstract— It is our view that the state of the art in constructing a large collection of graph algorithms in terms of linear algebraic operations is mature enough to support the emergence of a standard set of primitive building blocks. This paper is a position paper defining the problem and announcing our intention to launch an open effort to define this standard.

Presented at HPEC 2013, Waltham, MA

GraphBLAS Forum

FFRDCs	Industry	Academia
• MIT/Lincoln Labs*	• Intel* †	• UC Santa Barbara*, Davis †, Berkeley
• Lawrence Berkeley NL* †	• IBM †	• Georgia Tech*
• CMU/Software Engineering Institute †	• NVIDIA	• Karlsruhe (KIT)*
• Pacific Northwest NL	• Hauwei	• CMU
• Sandia NL	• Reservoir Labs	• Indiana U.
	• Galois	• MIT
	• Mellanox	• U. Washington
	• and others	• and others

GraphBLAS Primitives

Operation	Description
mxm, mxv, vxm	Perform matrix *multiplication* (e.g., breadth-first traversal)
eWiseAdd, eWiseMult	Element-wise *addition* and *multiplication* of matrices (e.g., graph union, intersection)
extract	Extract a sub-matrix from a larger matrix (e.g., sub-graph selection)
assign	Assign to a sub-matrix of a larger matrix (e.g., sub-graph assignment)
apply	Apply *unary function* to each element of matrix (e.g., edge weight modification)
reduce	*Reduce* along columns or rows of matrices (vertex degree)
transpose	Swaps the rows and columns of a sparse matrix (e.g., reverse directed edges)
buildMatrix	Build an matrix representation from row, column, value tuples
extractTuples	Extract the row, column, value tuples from a matrix representation

Specification for a set of base operations from which complex graph algorithms can be built in software

GraphBLAS in the Community

- Reference Implementations
  - SuiteSparse:GraphBLAS (latest version) can be downloaded from here.
  - IBM GraphBLAS
- Projects developing implementations of the GraphBLAS
  - MPI/C++ Combinatorial BLAS (CombBLAS)
  - Java Graphulo
  - Matlab/Octave D4M
  - GPI (IBM) and IBM GraphBLAS
  - GraphPad (Intel)
  - GraphBLAS Template Library (SEI/Indiana)
  - SuiteSparse Graph BLAS (Texas A&M)
  - GraphBLAST (UC Davis and LBNL)
- Graph analysis systems that integrate GraphBLAS
  - RedisGraph: A Graph Database Module for Redis
  - pggraphblas: High Performance Graph Processing with PostgreSQL

SuiteSparse Performance

Single-threaded performance: Sparse DNN

Single threaded performance (rate, in billion edges/sec, higher is better)												
Neurons:	1K			4K			16K			64K		
Layers:	120	480	1920	120	480	1920	120	480	1920	120	480	1920
MATLAB	2	2	2	2	2	2	2	2	2	2	2	1.5
in SuiteSparse:GraphBLAS:												
DGX+C	11	16	19	10	13	14	5	6	6	4	5	5
DGX+M	10	14	16	10	13	14	5	6	6	4	4	4

GraphBLAS about 2x to 8x faster than pure MATLAB with a single thread. Little performance lost when using GraphBLAS via its MATLAB interface (DGX+M).

Parallel threaded performance: Sparse DNN

OpenMP and/or MPI performance (rate; higher is better)												
DGX+C	**153**	**240**	**275**	100	127	143	75	83	88	60	68	65
DGX+M	86	101	110	95	123	129	65	75	77	59	62	63
Power8	111	139	150	125	164	183	102	112	114	**105**	**109**	**113**
Power9	119	177	190	**139**	**174**	**186**	**105**	**116**	101	90	-	-
Xeon	58	68	71	115	140	150	99	115	**120**	81	89	-

Highest performance in bold. GraphBLAS is 60x to 120x faster than the pure MATLAB reference solution, on the same platform (DGX). Normally little performance lost when using GraphBLAS via its MATLAB interface (DGX+M).

GraphBLAS is rapidly gaining traction in the community and will be an ideal platform for Lucata

T. Davis, et. al., HPEC 2019, Waltham, MA

GraphBLAS Application - Graph Databases

- What is a graph database?
  - A database built around a graph structure
  - Nodes are indexed for fast initial lookup
  - Property Graph
    - Each node/edge is uniquely identified
    - Each node has a set of incoming and outgoing edges
    - Each node/edge has a collection of properties
    - Each edge has a label that defines the relationship between its two nodes
- Comparing with Relational DB
  - Pros:
    - Schema flexibility: unstructured data, dynamic relationships
    - More intuitive querying
    - Avoids "join bombs"
    - Local hops are not a function of total nodes
  - Cons:
    - Not always advantageous
    - Query languages are not unified
- Examples include RedisGraph, Neo4j, and others

P. Nellaru, B. Naik, E. Liu, B. Koo

Takeaways

- The GraphBLAS API provides a useful abstraction for building complex systems and applications to support graph workloads
- GraphBLAS implementations can be used for both HPC applications and also to support large-scale graph database deployments