Graphite: A NUMA-aware HPC System for Graph Analytics Based on a new MPI∗X Parallelism Model

Mohammad Hasanzadeh Mofrad, Rami Melhem,

Yousuf Ahmad, and Mohammad Hammoud

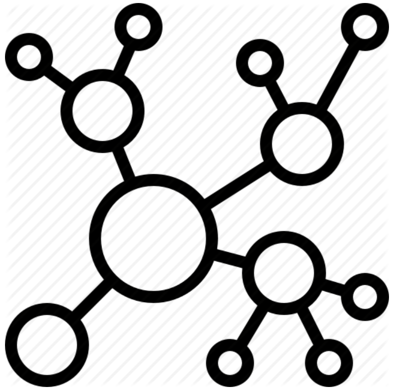
Github Repository: https://github.com/hmofrad/Graphite

Motivation: Big Graph Analytics

- Graphs are ubiquitous
  - **Google** PageRank
  - **Uber** routing's engine
  - **Amazon** item recommendation
  - **Facebook** friend suggestion

Background: The Duality Between Graphs and Sparse Matrices

- The relational data represented by graphs is often **hyper-sparse**
- Linear algebra primitives backed by sparse matrix formats are new alternatives to their graph-based versions

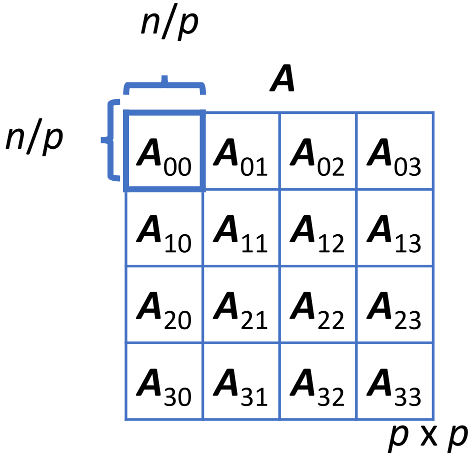


	Graph Theory	**Linear Algebra**
Representation	Graph	Adjacency Matrix
Data structure	Adjacency List	Compressed Sparse Format, e.g., CSC
Primitive	Fan-in/ Fan-out Operations	Sparse Matrix–Vector (SpMV) with GAS[1]
Scalability	**Graph Partitioning**	**Sparse Matrix Partitioning**

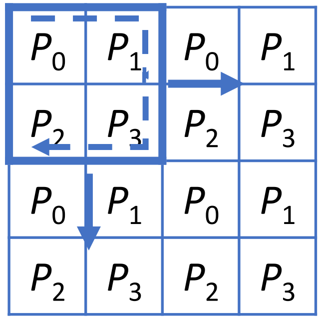
1. Gather, Apply, and Scatter (GAS) abstraction

2D Process-based Matrix Partitioning and Placement for scalability (e.g., p=4)

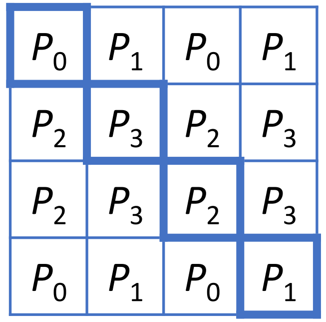
- **Processed-based 2D partitioning**: Given p processes, a matrix is partitioned into a pxp grid of tiles
  - Each process **computes** p tiles
- **2D cyclic placement (GraphPad, IPDPS 16)**: Given p processes, $\sqrt{p}$ processes are placed in each row/column group
  - Process **communication** is not part of the placement
- **2D Staggered partitioning (LA3, VLDB 18)**: Like 2D cyclic, however, unique processes are placed in diagonal tiles
  - Process **communication** is democratized by diagonal processes (O(p^2) time)



2D-process-based partitioning



2D-Cyclic Placement

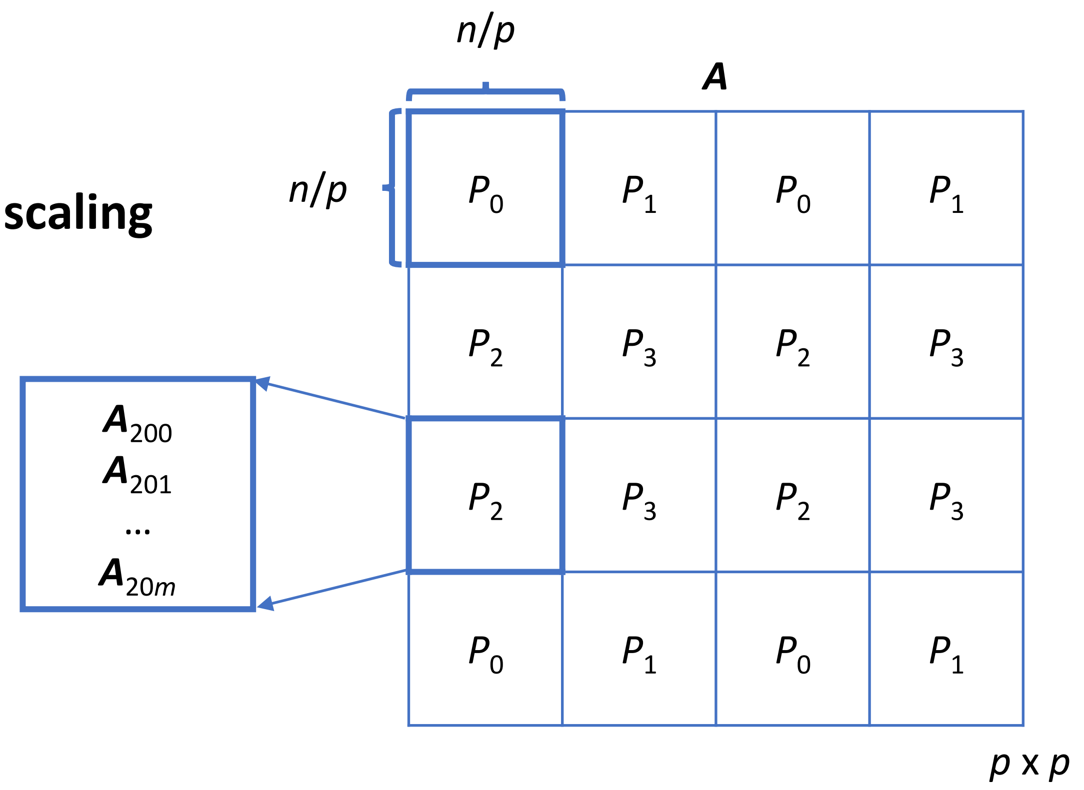


2D-Staggered Placement

MPI+X Parallelism Model for Multicore HPC Systems (e.g., p=4)

- **MPI+X parallelism model** utilizes
  - MPI to achieve **horizontal scaling**
    - One MPI process per machine
  - A threading library (X), e.g, Pthread, to achieve **vertical scaling**

- **MPI+X** uses process-based partitioning & placement

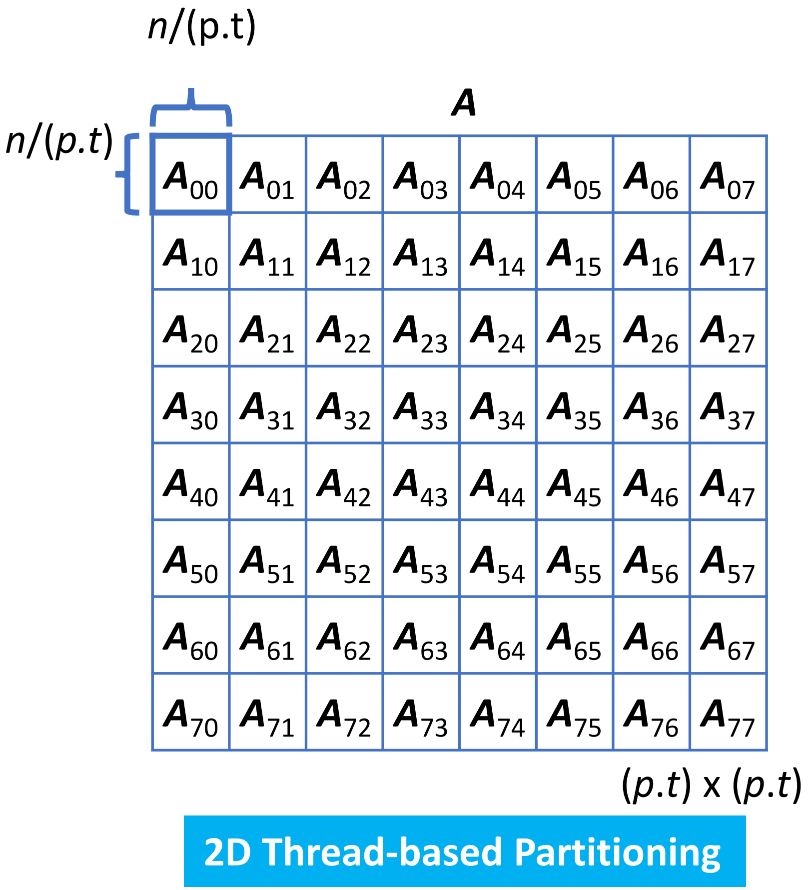


MPI+X with Processed-based 2D-Staggered

m is the number of sub-partitions

The New 2D Thread-based Matrix Partitioning and Placement for scalability (e.g., p=4, t=2)

- **Thread-based 2D partitioning**: Given p processes & t threads, a matrix is partitioned into a $(p.t)$x$(p.t)$ grid of tiles
  - Each global thread **computes** t tiles



The New 2D Thread-based Matrix Partitioning and Placement for scalability (e.g., *p*=4, *t*=2)

- **Thread-based 2D partitioning**: Given *p* processes & *t* threads, a matrix is partitioned into a (*p*.*t*)x(*p*.*t*) grid of tiles
  - Each global thread **computes** *t* tiles

- **New Thread-based 2D Staggered placement (2DT-Staggereed)**:
  - *p*.*t* global threads; *t* local threads per process
  - Thread **communication** is democratized by diagonal threads (O(2(*p*.*t*) time)

A

T_0	T_1	T_0	T_1	T_0	T_1	T_0	T_1
T_2	T_3	T_2	T_3	T_2	T_3	T_2	T_3
T_4	T_5	T_4	T_5	T_4	T_5	T_4	T_5
T_6	T_7	T_6	T_7	T_6	T_7	T_6	T_7
T_2	T_3	T_2	T_3	T_2	T_3	T_2	T_3
T_4	T_5	T_4	T_5	T_4	T_5	T_4	T_5
T_6	T_7	T_6	T_7	T_6	T_7	T_6	T_7
T_0	T_1	T_0	T_1	T_0	T_1	T_0	T_1

(*p*.*t*) x (*p*.*t*)

2D Thread-based Placement (Global Thread Ids)

The New MPI*X Parallelism Model (e.g., p=4 and t=2)

- **MPI*X parallelism model** utilizes
  - Threads to achieve **diagonal scaling**
    - t global threads are grouped together to form a process
  - One MPI process per CPU socket and one thread per core
    - NUMA-aware

A

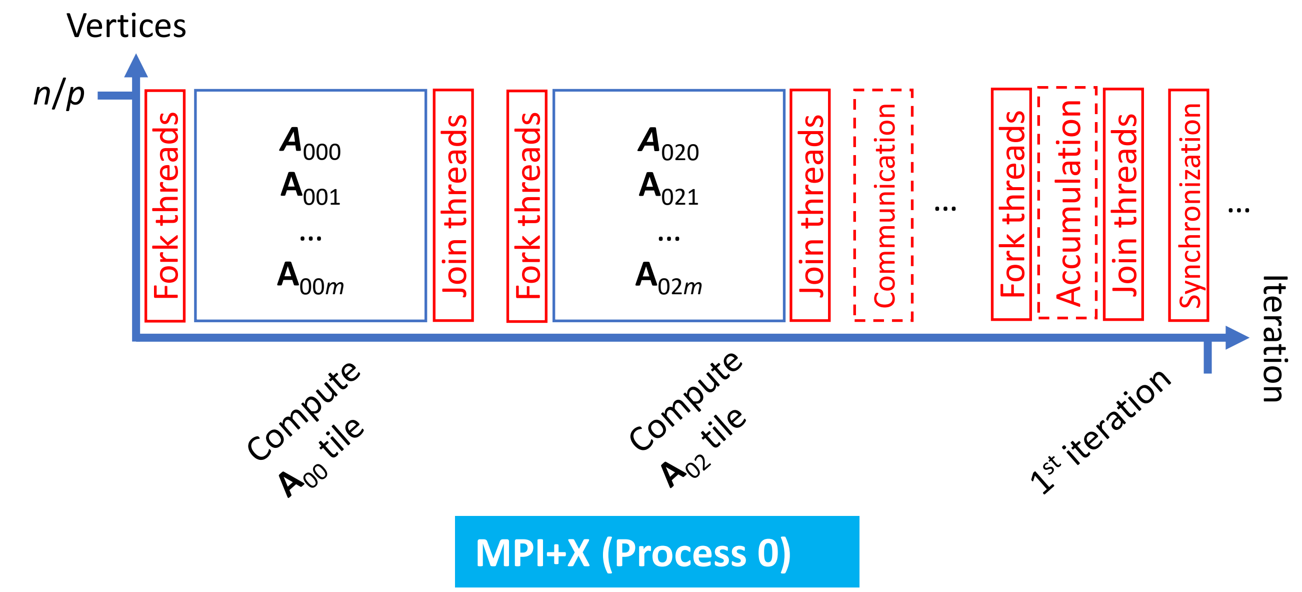
P_0T_0	P_1T_0	P_0T_0	P_1T_0	P_0T_0	P_1T_0	P_0T_0	P_1T_0
P_2T_0	P_3T_0	P_2T_0	P_3T_0	P_2T_0	P_3T_0	P_2T_0	P_3T_0
P_0T_1	P_1T_1	P_0T_1	P_1T_1	P_0T_1	P_1T_1	P_0T_1	P_1T_1
P_2T_1	P_3T_1	P_2T_1	P_3T_1	P_2T_1	P_3T_1	P_2T_1	P_3T_1
P_2T_0	P_3T_0	P_2T_0	P_3T_0	P_2T_0	P_3T_0	P_2T_0	P_3T_0
P_0T_1	P_1T_1	P_0T_1	P_1T_1	P_0T_1	P_1T_1	P_0T_1	P_1T_1
P_2T_1	P_3T_1	P_2T_1	P_3T_1	P_2T_1	P_3T_1	P_2T_1	P_3T_1
P_0T_0	P_1T_0	P_0T_0	P_1T_0	P_0T_0	P_1T_0	P_0T_0	P_1T_0

$(p.t) \times (p.t)$

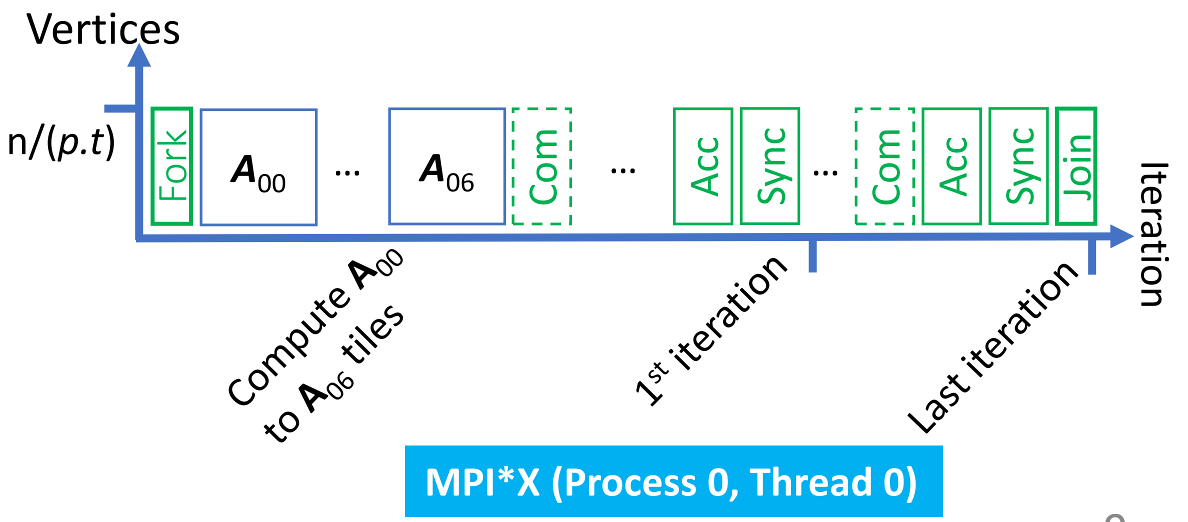
2D Thread-based Placement (Local Thread Ids)

MPI+X Parallelism Model (e.g., *p*=4) Versus the New MPI*X Parallelism Model (e.g., *p*=4 and *p*=2)

- MPI+X uses process-based partitioning and hence
  - When computing each tile, multiple threads are forked & joined
  - Only MPI processes are the communication endpoints
  - Synchronization points are designed for MPI processes



- MPI*X uses thread-based partitioning and so
  - Threads stay alive
  - Communication is done by threads (better overlapping)
  - Synchronization is done by threads



The **Graphite** HPC Graph Analytics System

- **MPI*X**
  - Thread-based partitioning & placement
  - Diagonal scaling
- **NUMA-aware**
  - Computation (CPU and memory affinity)
  - Communication (MPI shared-memory transport)
- **Asynchronous** computation & communication
  - Broadcasting (log(p) time) when possible
- **TCSC** (Triply Compressed Sparse Column) (GraphTap, Cluster 19)

Experimental Settings

Graph Processing Systems						
	Parallelism Model	**Unit**	**Partitioning**	**NUMA-aware**	**Compression**	**Primitive**
Graphite	MPI*X	Threads	2DT-Staggered	CPU/Memory	TCSC	SpMSpV[2] GAS
LA3 (VLDB, 2018)	MPI+X	Processes	2D-Staggered	N/A	CSC[1]	SpMV GAS
GraphPad (IPDPS, 2016)	MPI+X	Processes	2D-Cyclic	N/A	CSC[2]	SpMV GAS
Gemini (USENIX, 2016)	MPI+X	Processes	1D-Row	Memory	CSC/CSR	Fan in/out

Graph Applications
PageRank (PR)
Single Source Shortest Path (SSSP)
Breadth First Search (BFS)
Connected Component (CC)

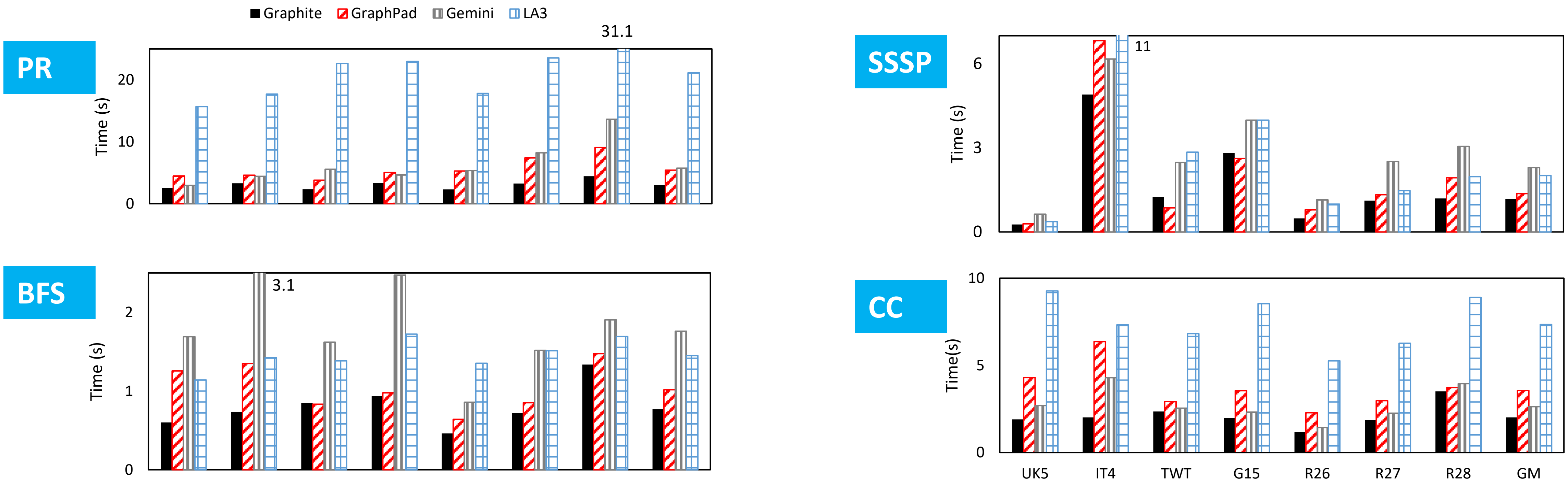
Node Specification	
CPU	28-core @ 2.6 GHZ
Memory	192 GB
OS	Linux
MPI	Intel
Network	Intel Omni-path Fabric

Datasets				
Graph	**|*V*|**	**|*E*|**	**Type**	**#Machines**
UK'05 (UK5)	39.4 M	0.93 B	Web	4
IT'04 (IT4)	41.2 M	1.15 B	Web	4
Twitter (TWT)	41.6 M	1.46 B	Social	8
GSH'15 (G15)	68.6 M	1.8 B	Web	8
UK'06 (UK6)	80.6 M	2.48 B	Web	16
UK Union (UKU)	133 M	5.5 B	Web	20
Rmat26 (R6)	67.1 M	1.07 B	Synthetic	4
Rmat27 (R27)	134 M	2.14 B	Synthetic	8
Rmat28 (R28)	268 M	4.29 B	Synthetic	16
Rmat29 (R29)	536 M	8.58 B	Synthetic	20

1. Compressed Sparse Column (CSC)
2. Compressed Sparse Row (CSR)

Results (Weak Scaling)

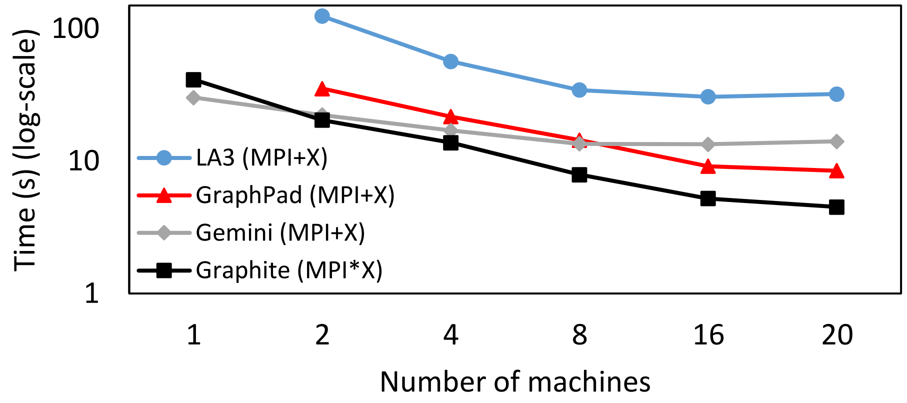
- Graphite is up to 3x faster than others which is due to its several good properties



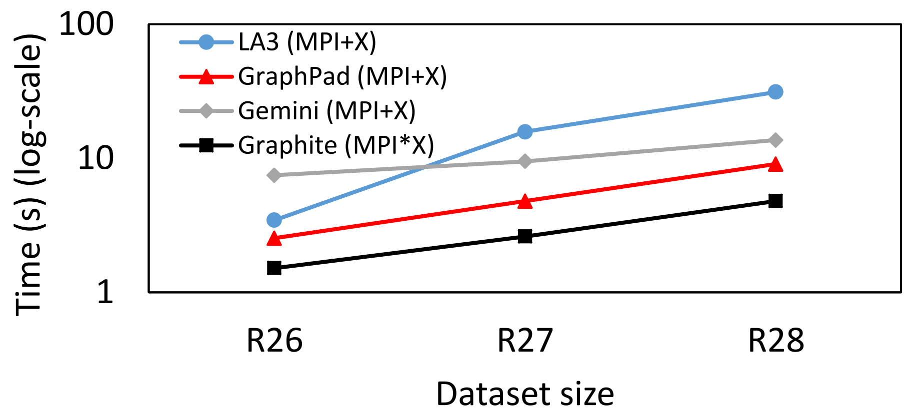
MPI*X versus MPI+X Strong Cluster and Data Scaling

- Graphite has superior cluster & data scalability because of **MPI*X**

- LA3, GraphPad, and Gemini are less scalable due to **MPI+X**



Cluster Scaling (PR)



Data Scaling (PR)

"I tried to come up with a joke about sparse matrices, but I'm just too dense"
@DocSparse

Stay Safe Everyone!

Thank you! :-)