Implementing Performance Portable Graph Algorithms Using Task-Based Execution

Ümit V. Çatalyürek
Georgia Institute of Technology & Amazon Web Services*

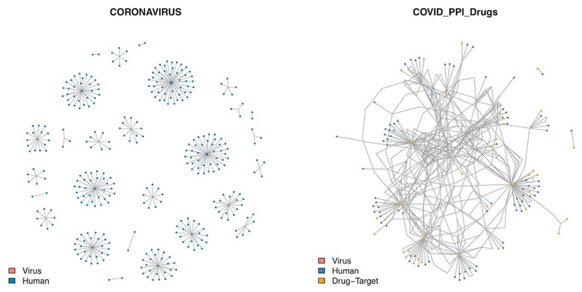
Joint work with

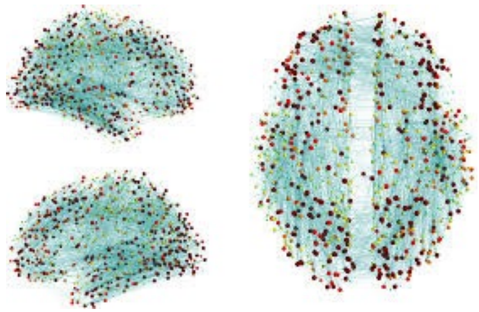
Abdurrahman Yaşar, Georgia Institute of Technology & NVIDIA

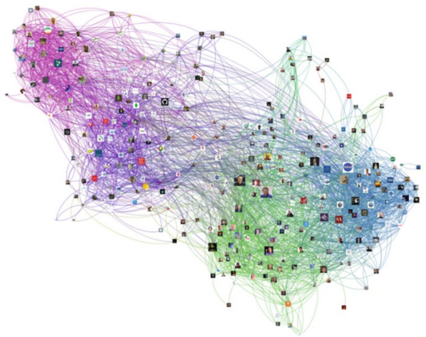
Sivasankaran Rajamanickam & Jonathan Berry, Sandia National Laboratories

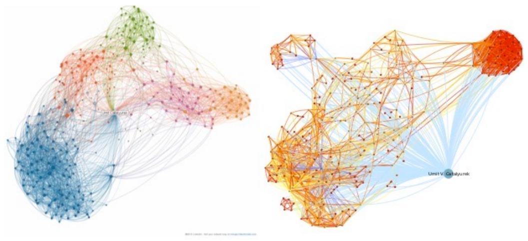
* This presentation describes work performed at Georgia Tech and is not associated with Amazon.

Graphs are Ubiquitous









Linkedin Ego-Net DBLP Ego-Net

They are growing. Up to billions of vertices and edges

Fast, efficient analysis is important and pervasive

Many graph processing frameworks have been proposed

Image credits:

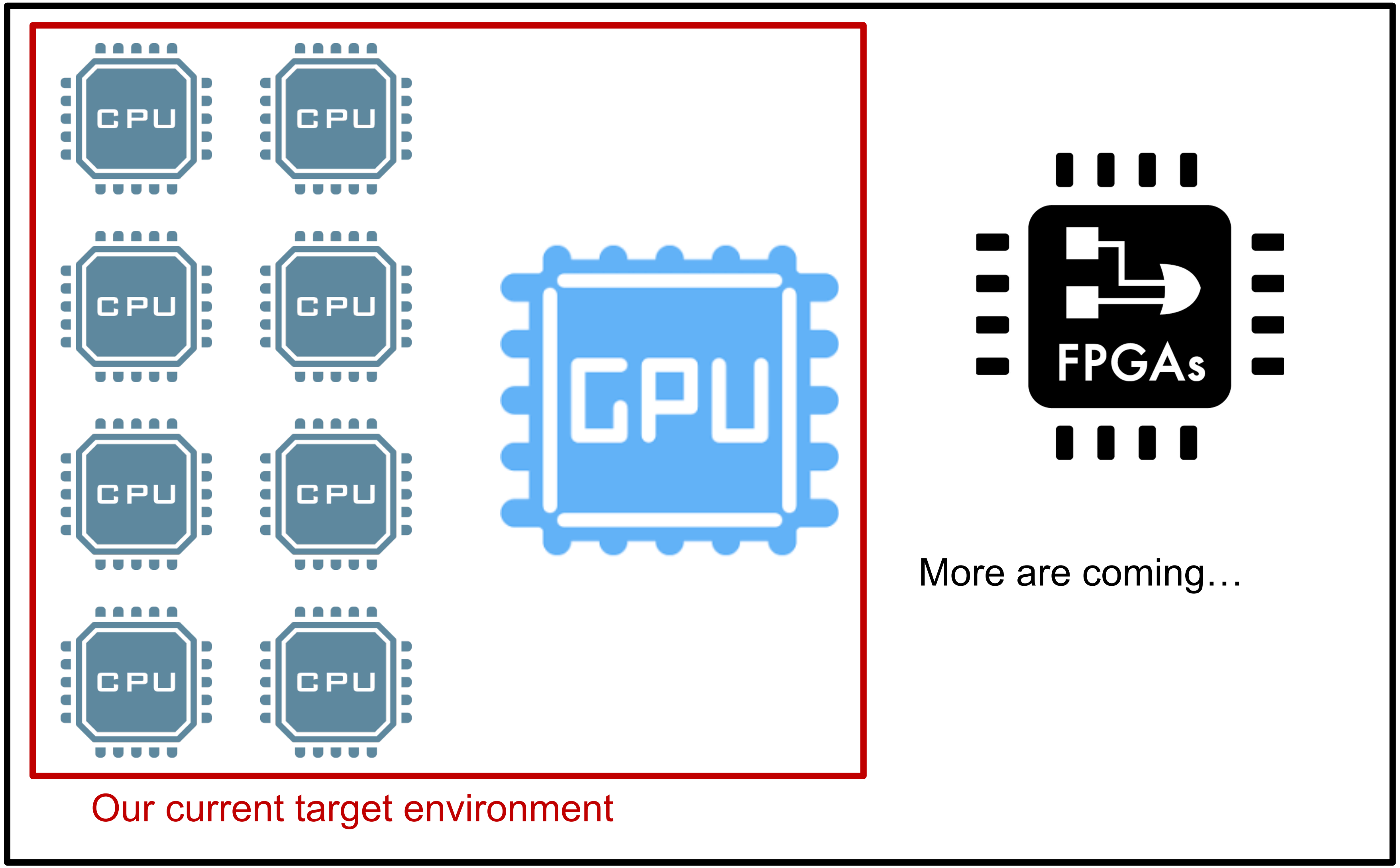
Jenn Caulfield, Social network vector illustration, 2018

Gerhard et al., Frontiers in Neuroinformatics 5(3), 2011

Albert-László Barabási/BarabasiLab 2019

Caleb Jonson, How to Visualize Your Twitter Network, 2014

Heterogeneous Systems are Here

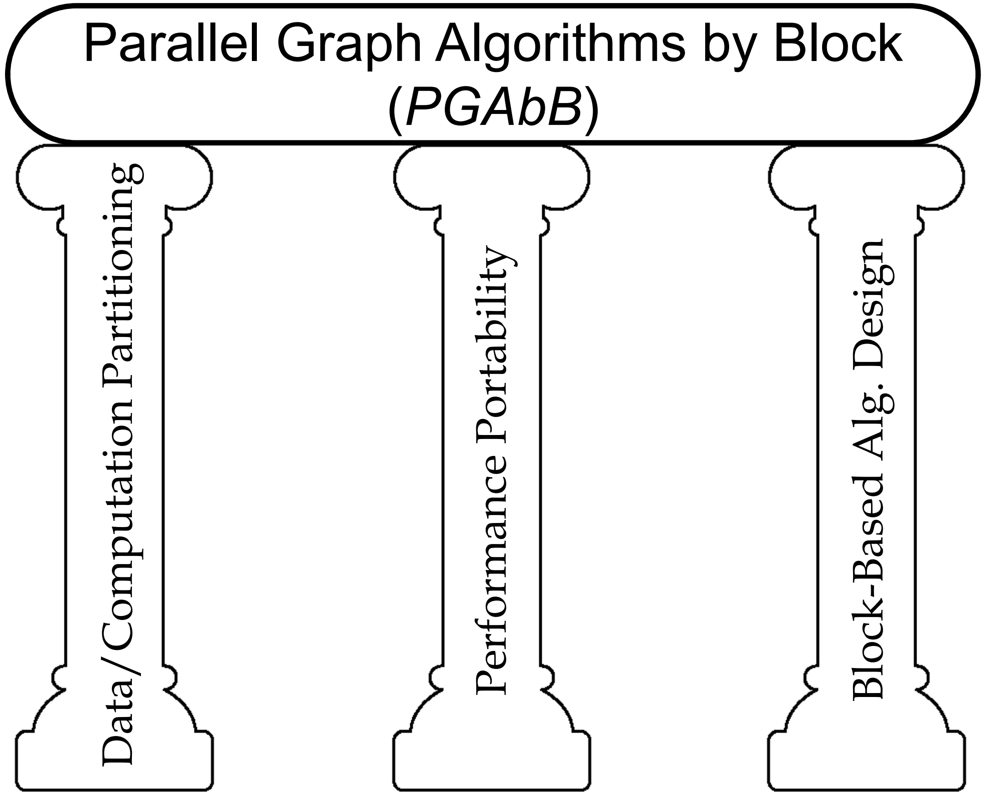


A Single Computing Node

The Crux

How can we develop efficient parallel graph algorithms that run well on **shared-memory and heterogeneous systems** as well as distributed-memory systems?

Block-based graph algorithms offer a good compromise between efficient parallelism and architecture agnostic algorithm design

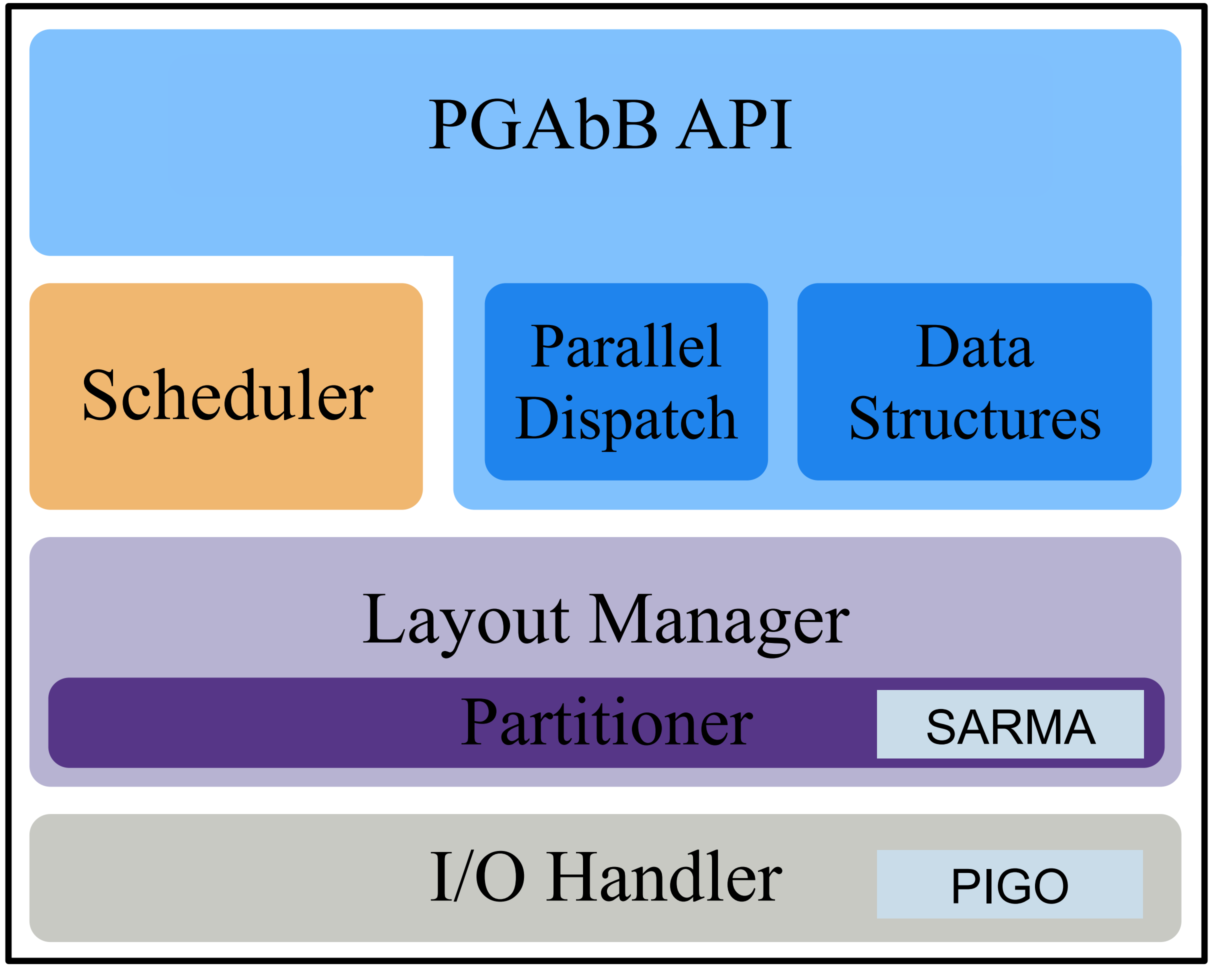


Design Goals of PGAbB

We have three design goals:

- An expressive programming model
- Execute graph kernel operations on different architectures.
  - Combine the results coming from different architectures
- Address major efficient parallel graph algorithm implementation challenges at behind the scenes.

System Overview



An Overview of PGAbB

https://github.com/GT-TDAlab/PIGO *https://github.com/GT-TDAlab/SARMA*

Algorithm Design Steps

Block List Composition → Attribute Assignment → Execution Handling → Kernel Development

$\mathcal{P}_G, \mathcal{P}_C$

Generic and Custom Block-List Generators.

PGAbB API

$\mathcal{I}_B$, $\mathcal{I}_A$, $\mathcal{E}$

Before Iter.

After Iter.

WorkEs tim.

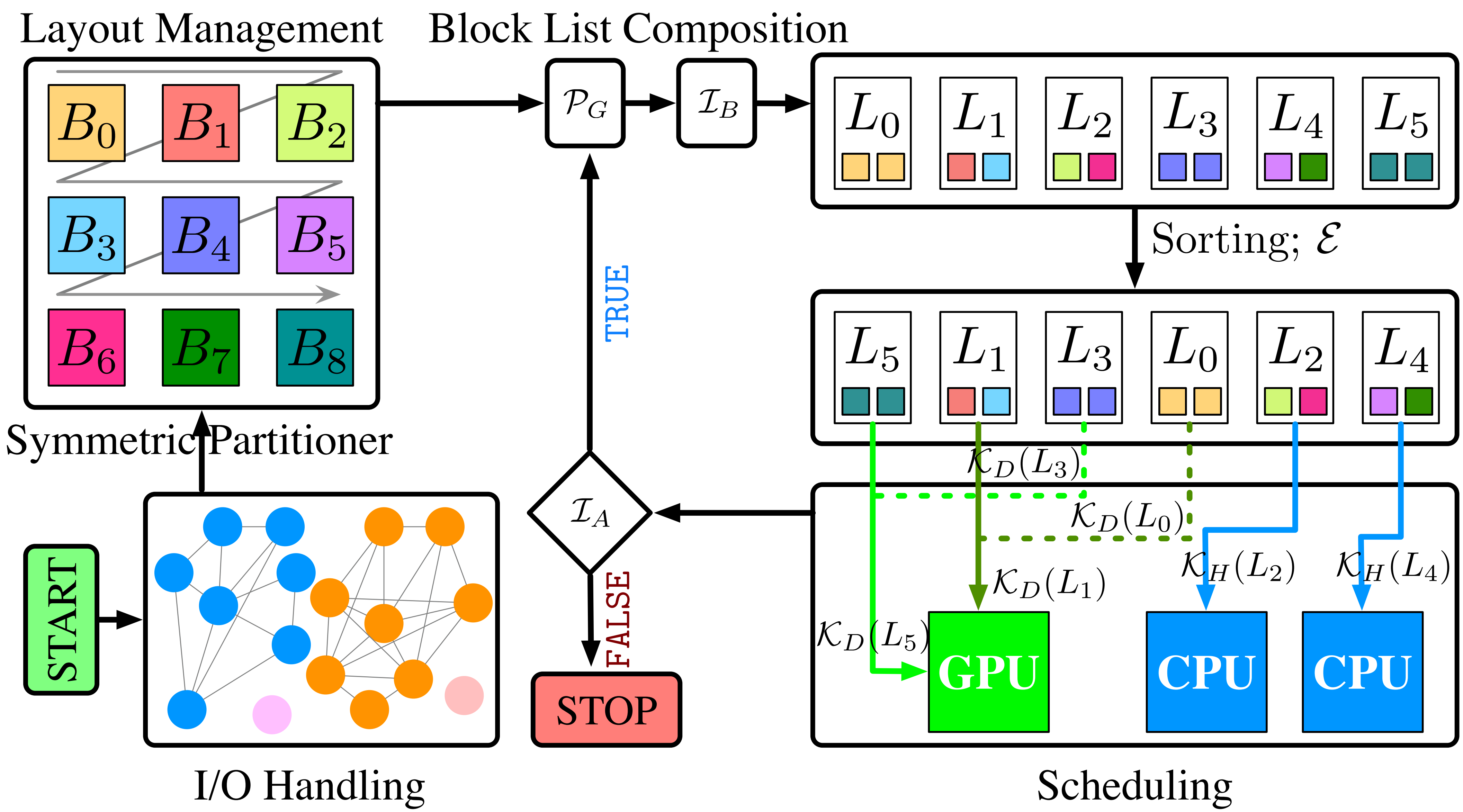
$\mathcal{K}_H, \mathcal{K}_D$

Host and Device Kernels

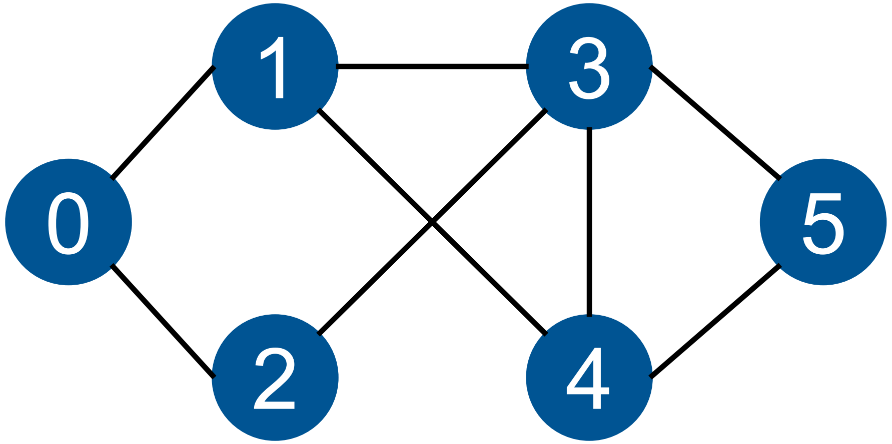
Required

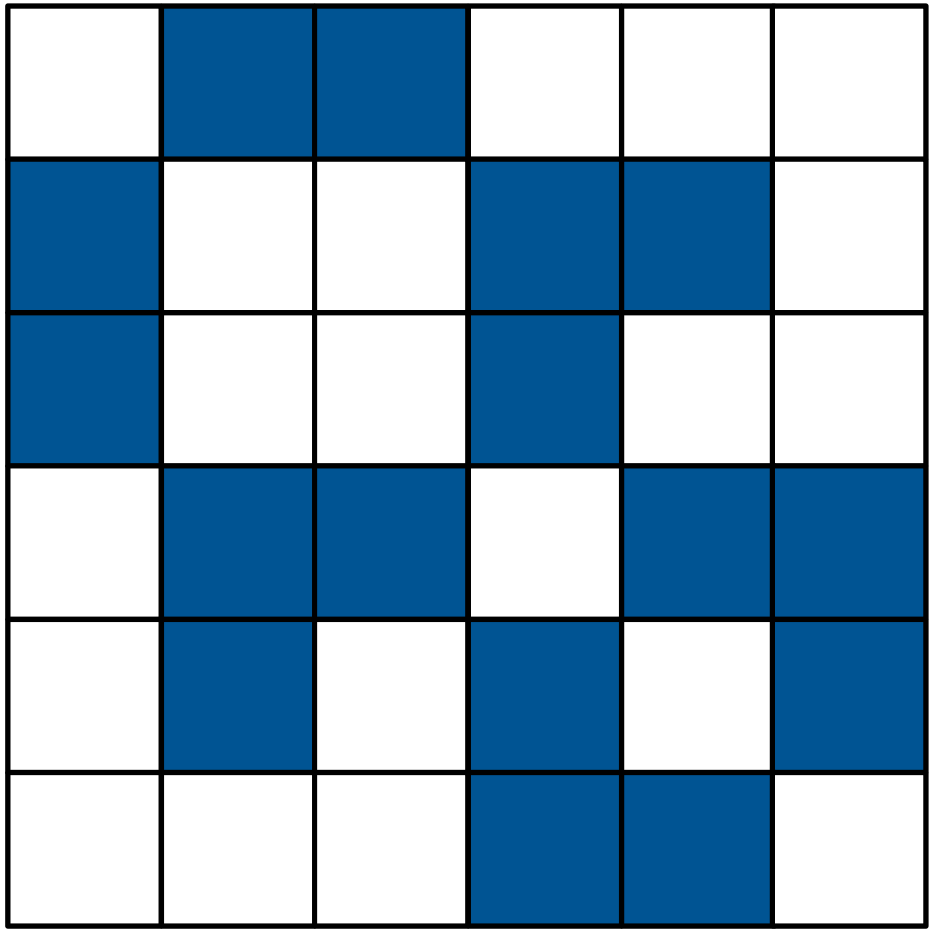
Optional

Execution Flow

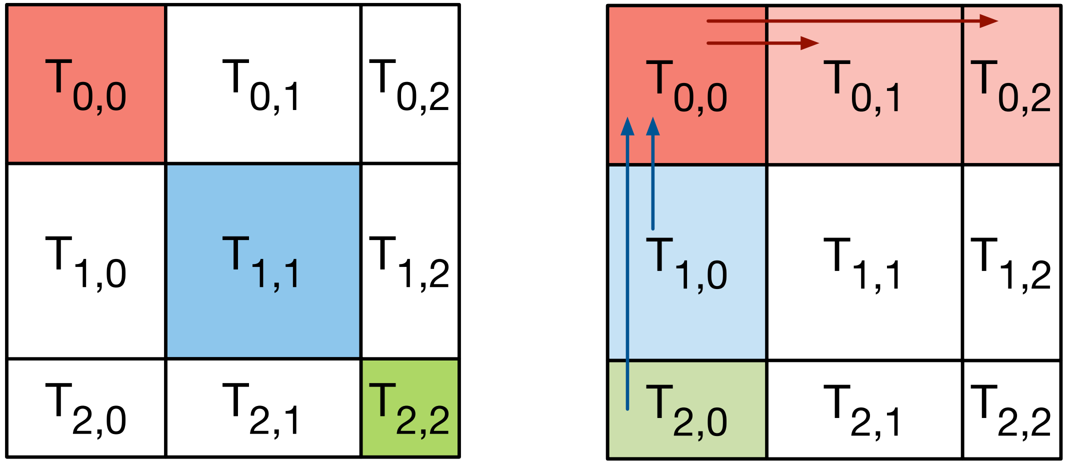


Toy Graph





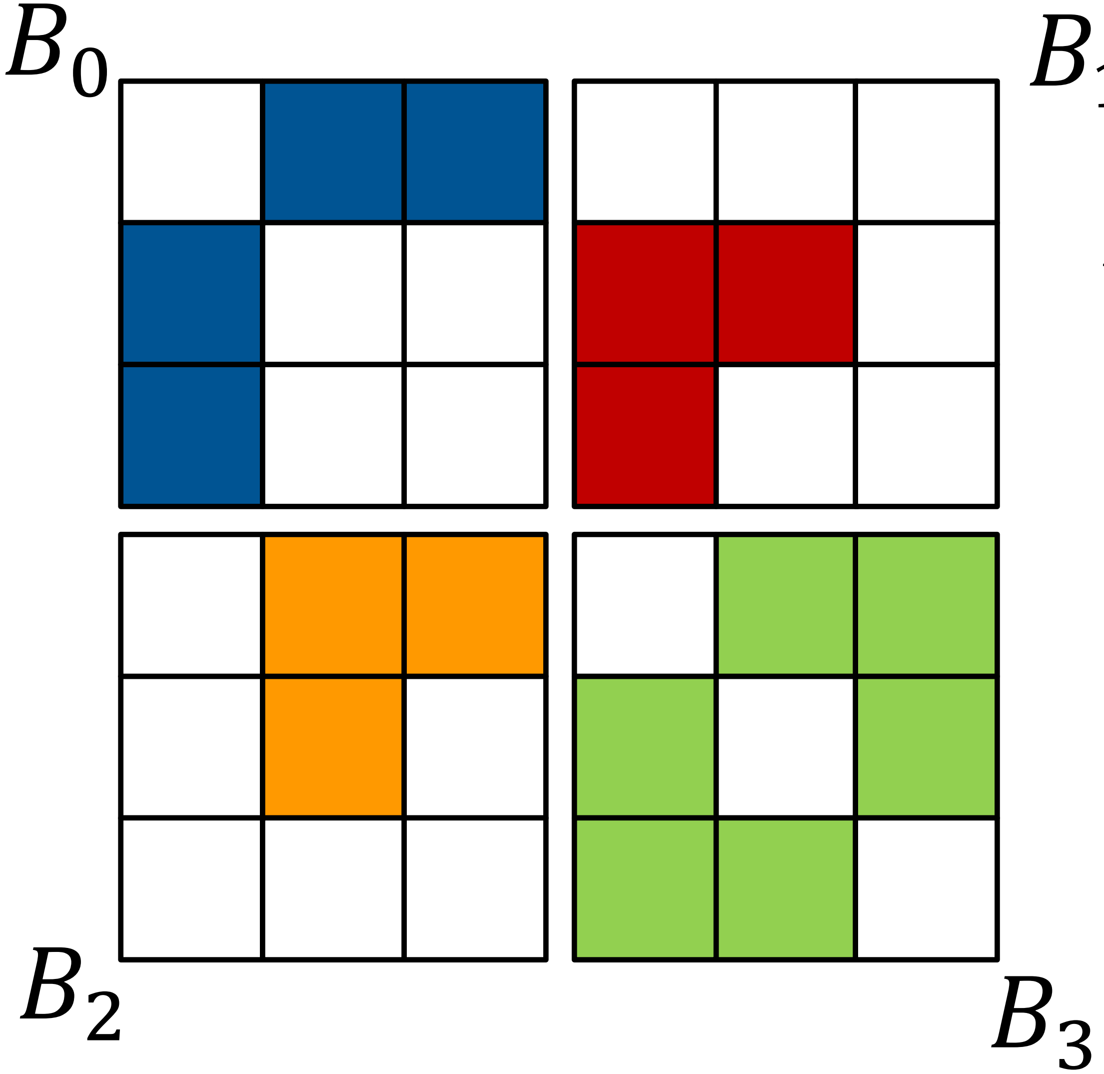
Symmetric Rectilinear Partitioning



A simple example

- Restricted rectilinear partitioning:
  - Can be obtained by aligning the same partition vector to rows and columns.
  - We showed this problem is NP-Complete too.
  - We proposed several heuristics and optimizations.
- PGAbB can be used with 1D and 2D partitioning. We will use 2D symmetric partitioning in this talk.

Block



A block (B_i) : Set of edges.

Graph, $G = \cup B_i$, and $\cap B_i = \emptyset$

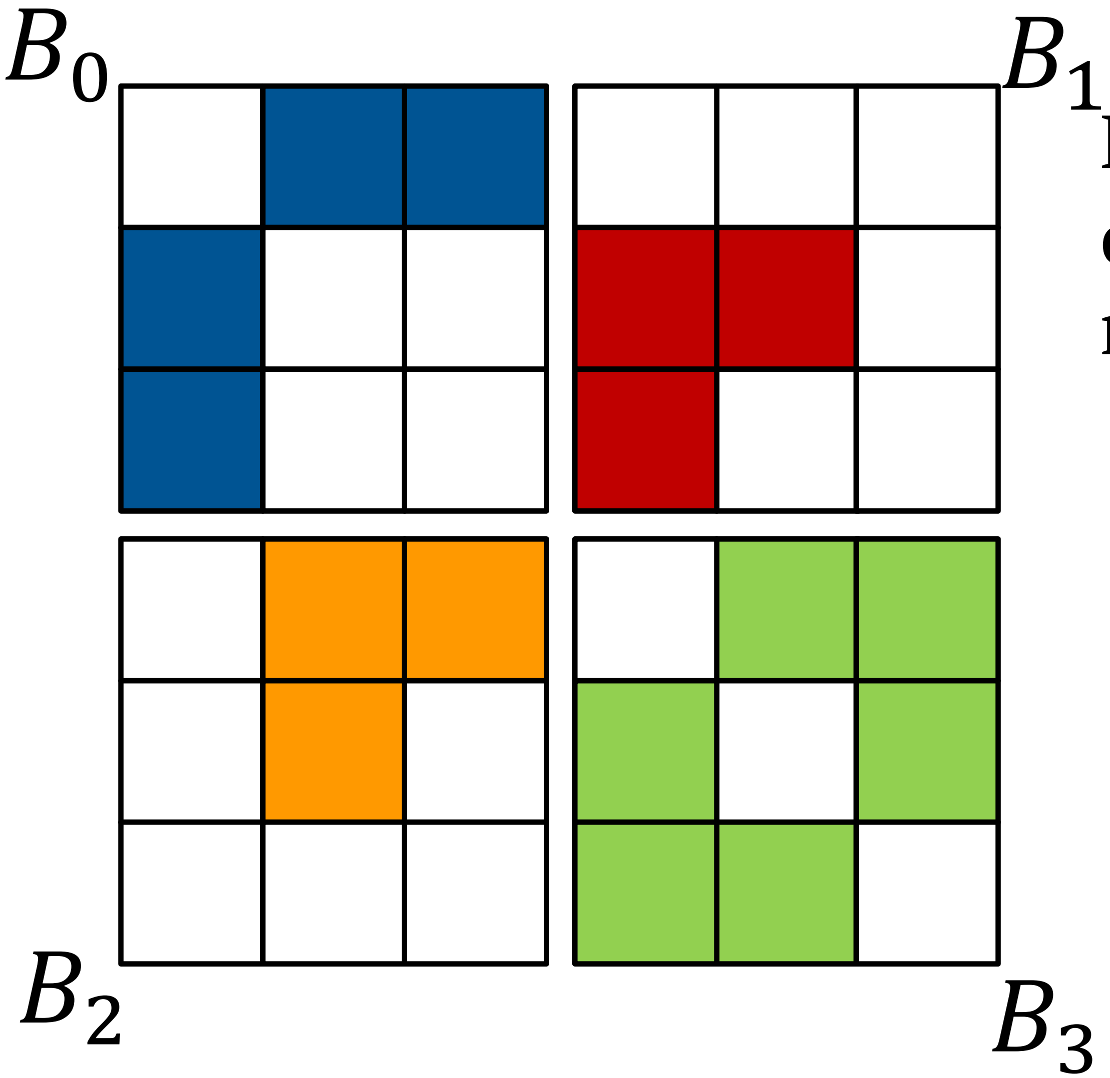
$B_0 = \{(0,1), (0,2), (1,0), (2,0)\}$

$B_1 = \{(1,3), (1,4), (2,3)\}$

$B_2 = \{(3,1), (3,2), (4,1)\}$

$B_3 = \{(3,4), (3,5), (4,3), (4,5), (5,3), (5,4)\}$

Block List



Block list ($\mathrm{L_j} = \langle B_i, B_1, .., B_k \rangle$) : list of ordered block references based on a rule.

$\mathrm{L}_0 = \langle B_0, B_1 \rangle$

$\mathrm{L}_1 = \langle B_1, B_2 \rangle$

Combined Example

$\mathrm{L}_0 = \langle B_0 \rangle$

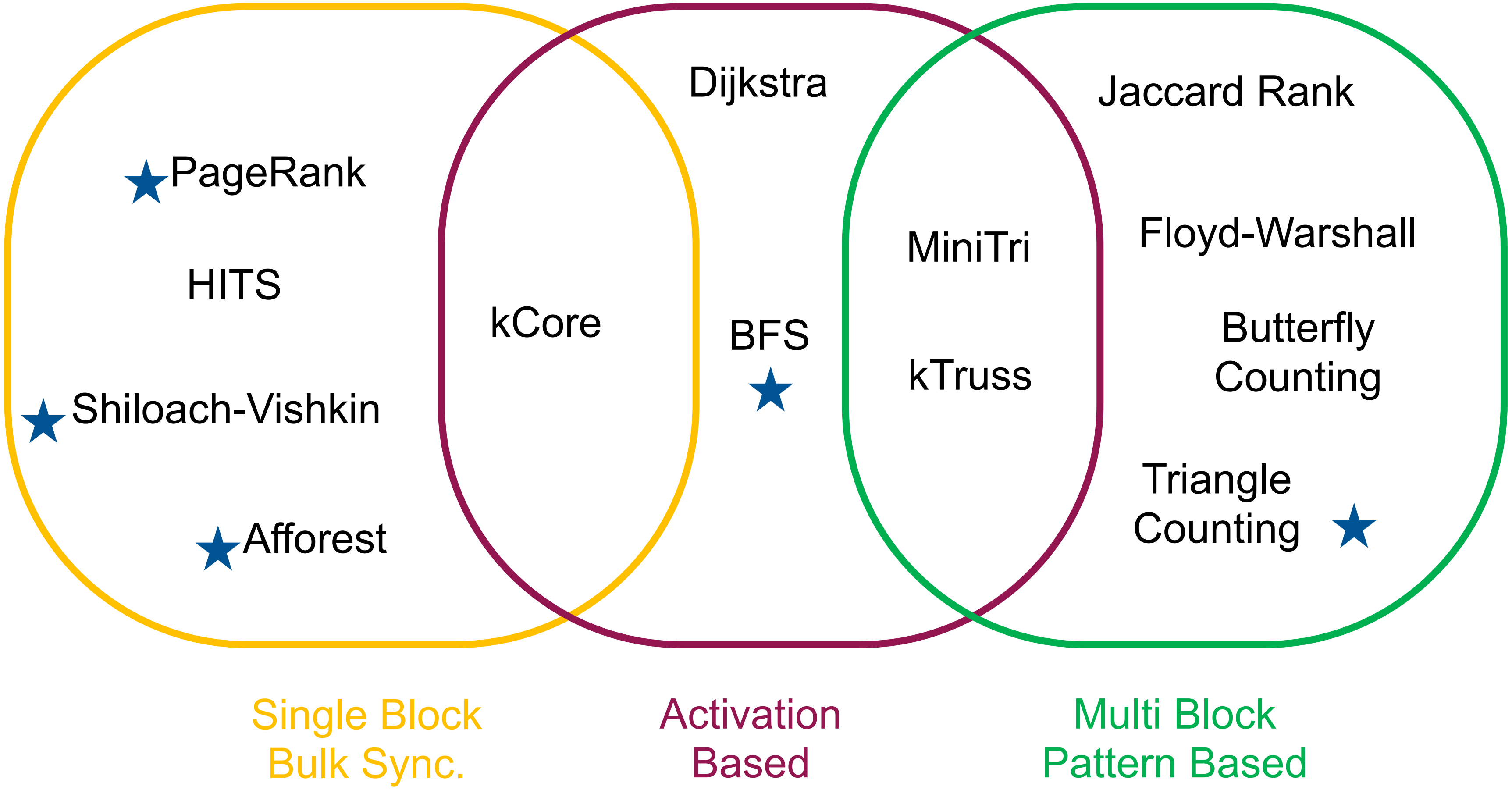
$\mathrm{L}_1 = \langle B_1, B_3 \rangle$

$\mathrm{L}_2 = \langle B_0, B_2, B_3 \rangle$

$\mathrm{L}_3 = \langle B_3 \rangle$

Generalized Example

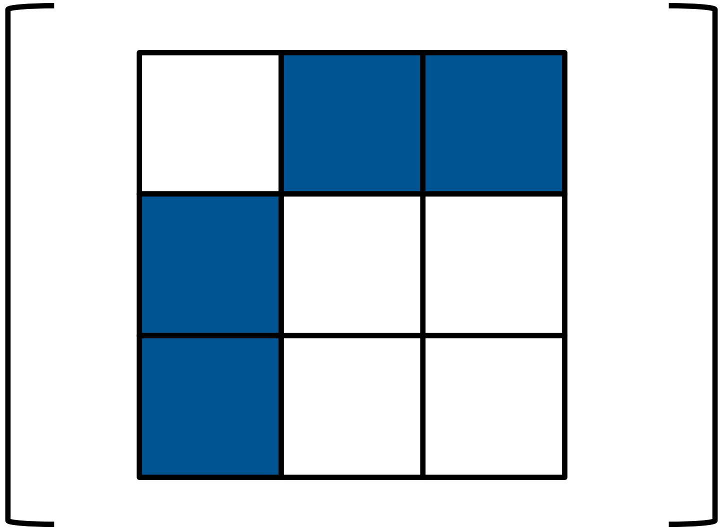
Categorizing Graph Algorithms



Kernel

A kernel is functor that takes a block list as input

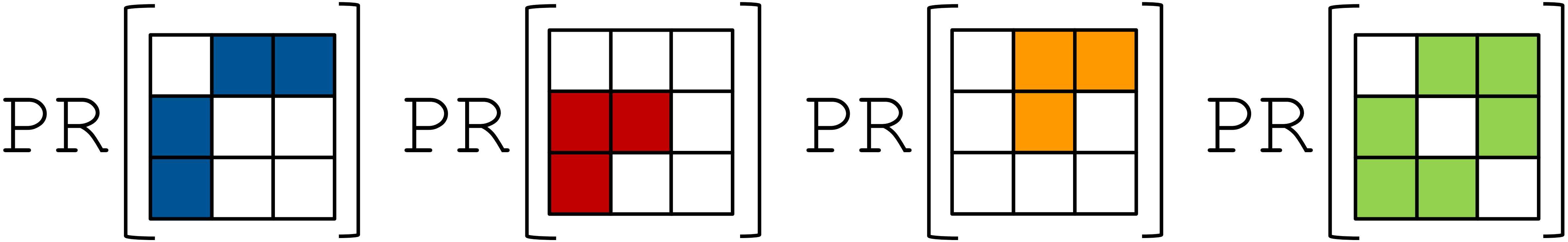
PR



$$PageRank = \bigcup_i \mathrm{PR}(\langle B_i \rangle)$$

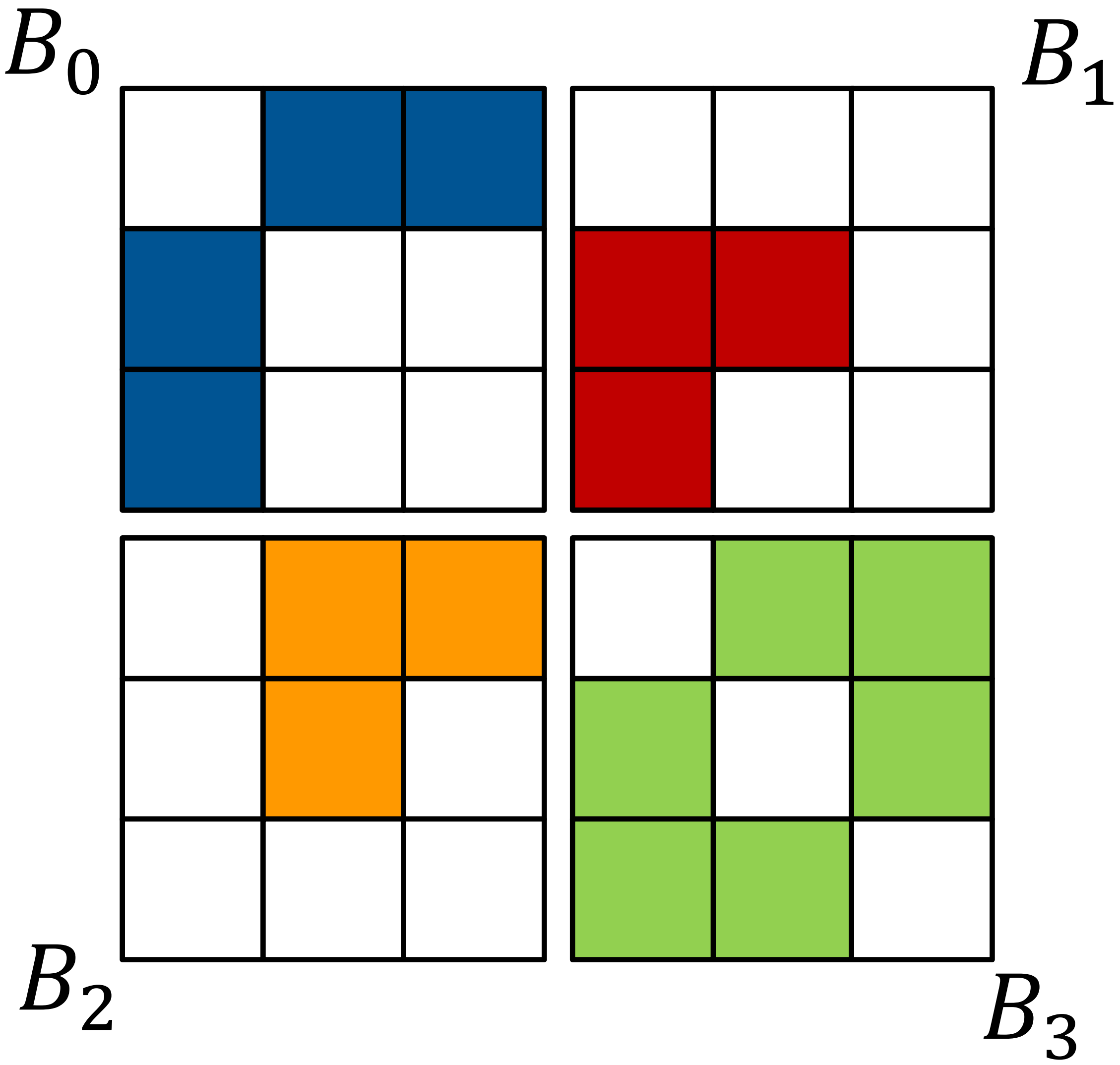
Task

A task, T_i, is defined with a kernel that operates on a block list.



T_0 T_1 T_2 T_3

Attributes



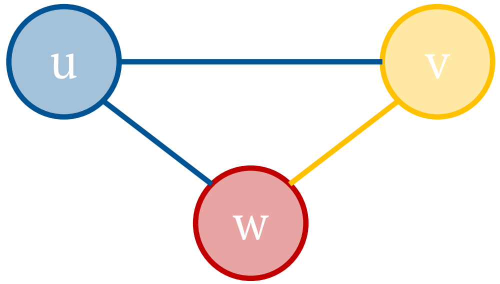
Vertex	Edge	Global
Diagonal blocks	Blocks	Custom
Ref. to Source and Destination	Self	Custom
Reduction Methods	Reduction Methods	Custom

Implemented Algorithms

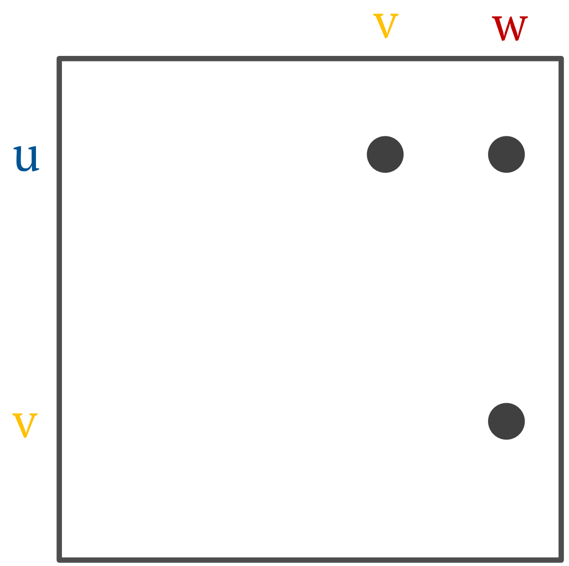
	Block-List	Attribute	Before Iter.	After Iter.	Host & Device Kernels
PageRank	Single-Block	Vertex	-	Check Err.	$Rank\ Sum \rightarrow Score\ Comp.$
Shiloach-Vishkin	Single-Block	Global: Array, counter	Reset Counter	Check counter	$Hook \rightarrow Link$
Afforest	Single-Block	Global: Array	-	-	$\text{Sample} \rightarrow Compress \rightarrow Connect \rightarrow Compress$
BFS	Activation	Global: Queues	-	Check Queue	Top-Down and/or Bottom-Up BFS
Triangle Counting	Multi-Block	Global: Counter var.	-	-	List Intersection

Triangle Counting Problem

Count mutually connected 3 vertices: u, v, w



Where u < v < w

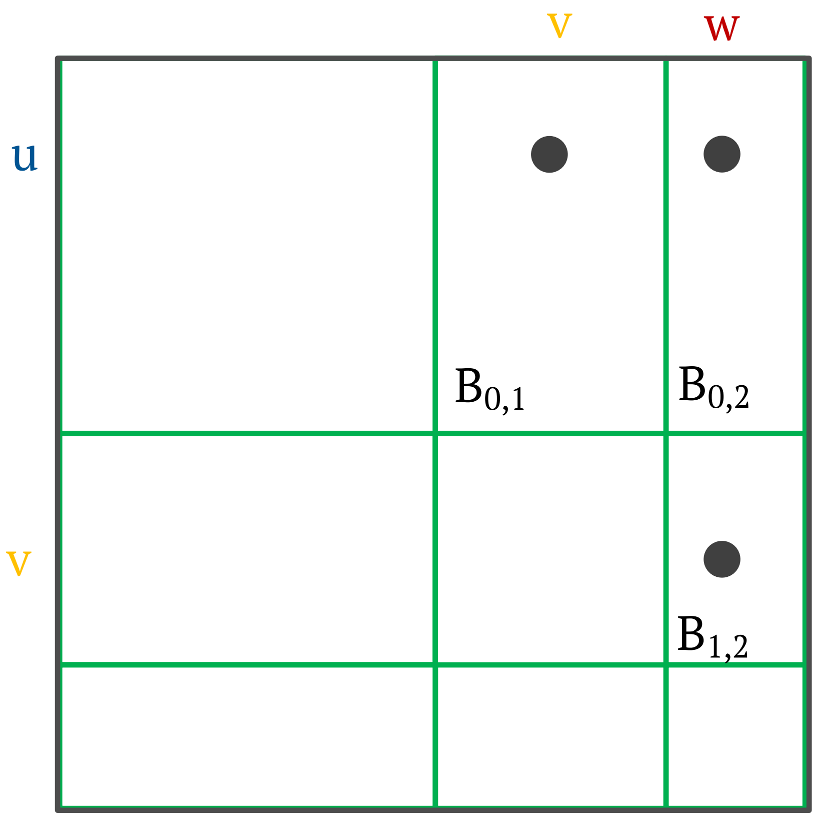


Triangle Counting Problem: Find the number of three-cycles (triangles) in an undirected graph G.

Important kernel which forms the core of;

- community detection,
- dense sub-graph discovery,
- k-truss decomposition,
- sub-graph isomorphism etc.

Partitioning and Task Construction



2D Partitioning

Cartesian Symmetric Rectilinear

(u,v) in $B_{0,1}$

(v,w) in $B_{1,2}$

(u,w) in $B_{0,2}$

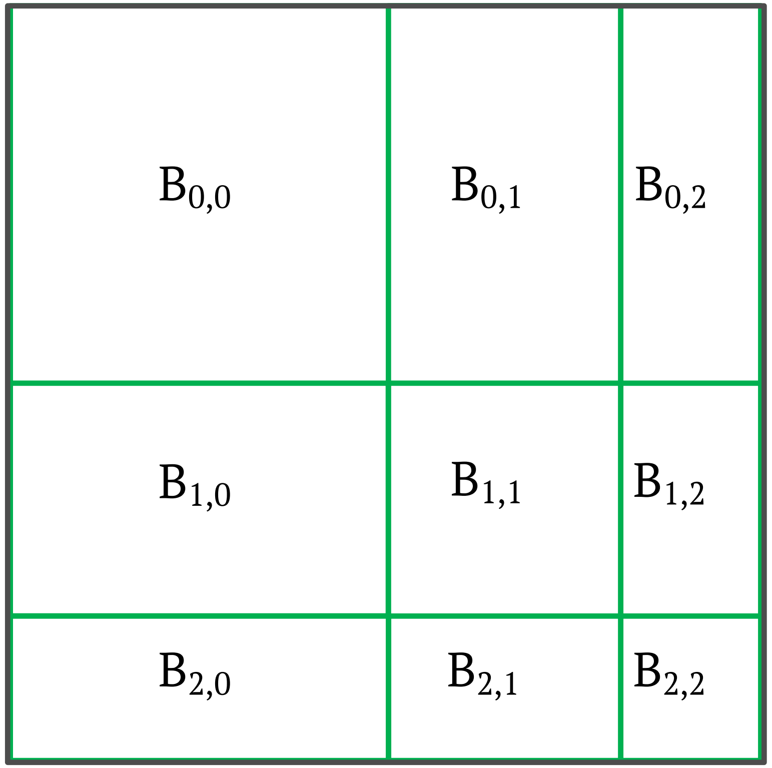
Block List: Triple of Blocks

$B_{i,j} - B_{j,k} - B_{i,k}$

$i \leq j \leq k$

How to Compose Task List

A Task: $LI(\langle B_{i,j}, B_{j,k}, B_{i,k} \rangle)$



Task	Workload	Order	Task	Workload	Order
$B_{0,0}\ B_{0,0}\ B_{0,0}$	13	4	$B_{0,2}\ B_{2,2}\ B_{0,2}$	15	2
$B_{0,0}\ B_{0,1}\ B_{0,1}$	3	9	$B_{1,1}\ B_{1,1}\ B_{1,1}$	14	3
$B_{0,0}\ B_{0,2}\ B_{0,2}$	12	6	$B_{1,1}\ B_{1,2}\ B_{1,2}$	12	5
$B_{0,1}\ B_{1,1}\ B_{0,1}$	9	8	$B_{1,2}\ B_{2,2}\ B_{1,2}$	19	1
$B_{0,1}\ B_{1,2}\ B_{0,2}$	11	7	$B_{2,2}\ B_{2,2}\ B_{2,2}$	20	0

Task list composition.

Workload estimation.

Sorting task list

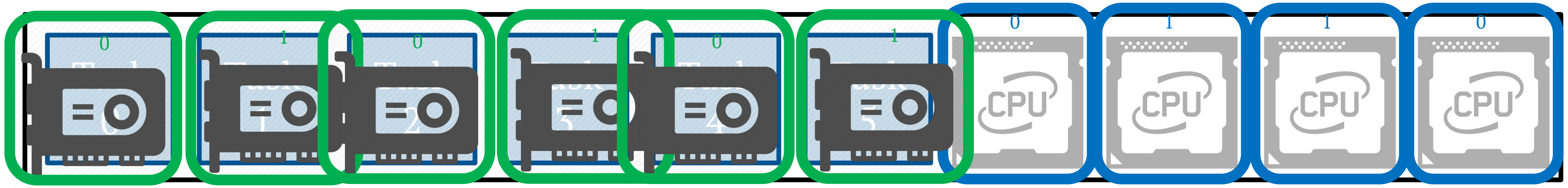
Hybrid Execution

Heavier Tasks: GPU

from heavier to lighter

Lighter Tasks: CPU

from lighter to heavier



Execution Queue

Sequential Execution Time Comparison in CPU

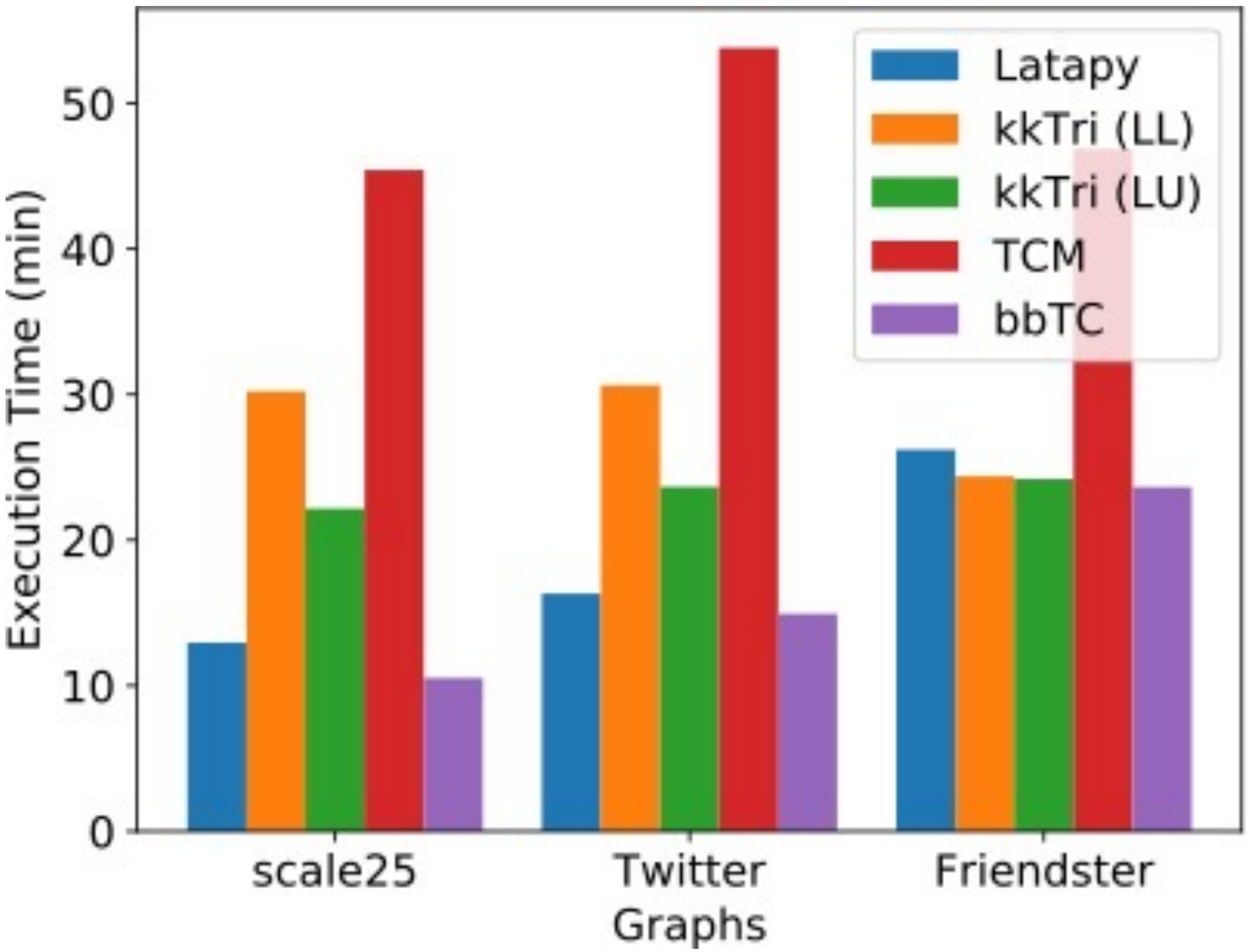
bbTC [TPDS'21] is available at
http://github.com/GT-TDAlab/bbTC

Latapy
Latapy; *"Main-memory triangle computations for very large (sparse (power-law)) graphs"*; TCS'08.

TCM
Shun and Tangwongsan; *"Multicore triangle computations without tuning"*; ICDE'15.

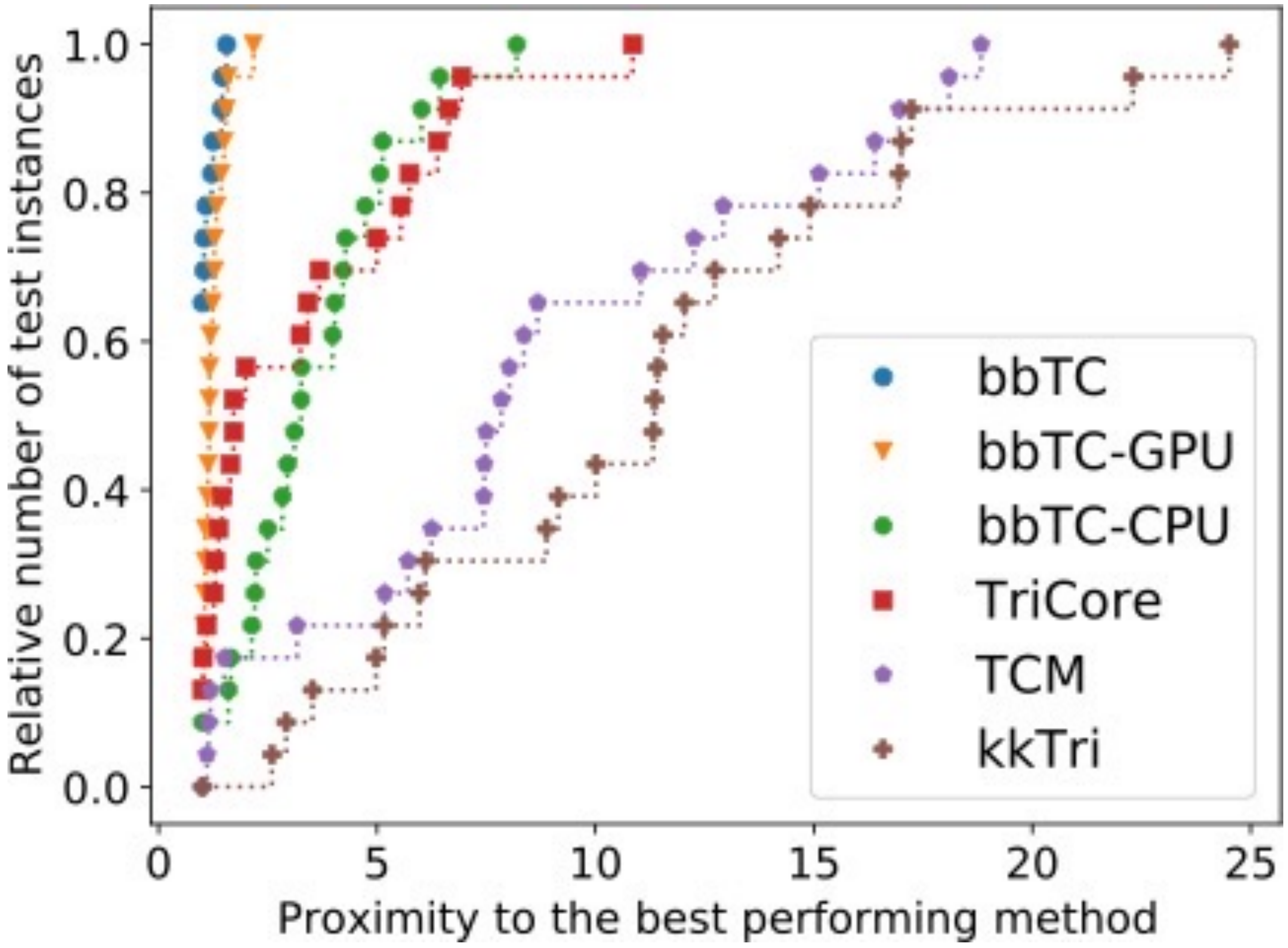
kkTri
Wolf et al.; *"Fast linear algebra-based triangle counting with kokkoskernels"*; HPEC'17.

TriCore (will be used next slide)
Liu et al.; *"Tricore: Parallel triangle counting on gpus"*; SC'18



Even sequential bbTC outperforms other algorithms in all graph instances.

Comparison with the state-of-the-art

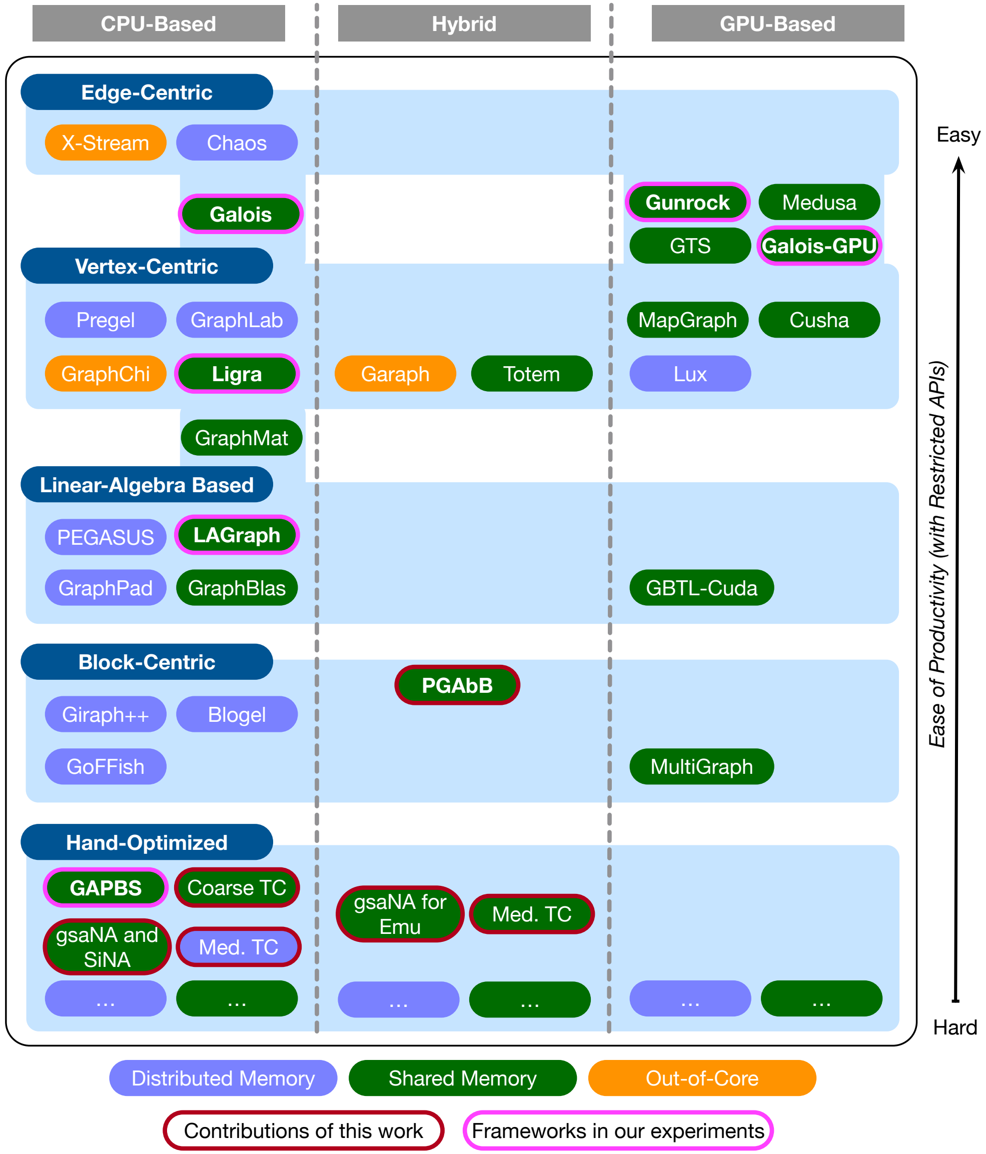


Running on a system with 2 x Power9 + 2 V100s

Even bbTC-GPU outperforms fastest GPU code TriCore*

*TriCore starts everything in GPU memory, and it is highly unstable: deviates up to 40%.

Related Work



Frameworks in Our Experiments

GAPBS: Beamer, et al., 2015. "The GAP benchmark suite.", ArXiV

Galois: Kulkarni, et al. 2007. *"Optimistic parallelism requires abstractions"*. PLDI

Ligra: Shun and Blelloch. 2013. *"Ligra: a lightweight graph processing framework for shared memory"*. PPoPP

LAGraph: Davis. 2019. *"Algorithm 1000: SuiteSparse: GraphBLAS: Graph algorithms in the language of sparse linear algebra"*, TOMS

Galois-GPU: Martin Burtscher, et al. 2012. *"A quantitative study of irregular programs on GPUs"*, IISWC

Gunrock: Wang, et al. 2016. *"Gunrock: A high-performance graph processing library on the GPU"*. PPoPP

Experimental Setup

- Power9 (2 x 16 x 4) CPUs with 2 Volta100 GPUs.
  - 320 GB Host Memory. 32 GB Device Memory.
  - CPU-GPU bandwidth: ~60GB/s
- Dataset: 44 graphs (real-world and synthetic), 100M-2.1B Edges
  - SuiteSparse, Konect, Snap
  - Converted to undirected and removed self-loops, duplicate edges.
  - In this talk: We are going to cover 7 of them in detail
- Algorithms: SV/LP, Best CC, PR, BFS, TC
- PGAbB: Kokkos at the backend with OpenMP (Host) and Cuda (Device)

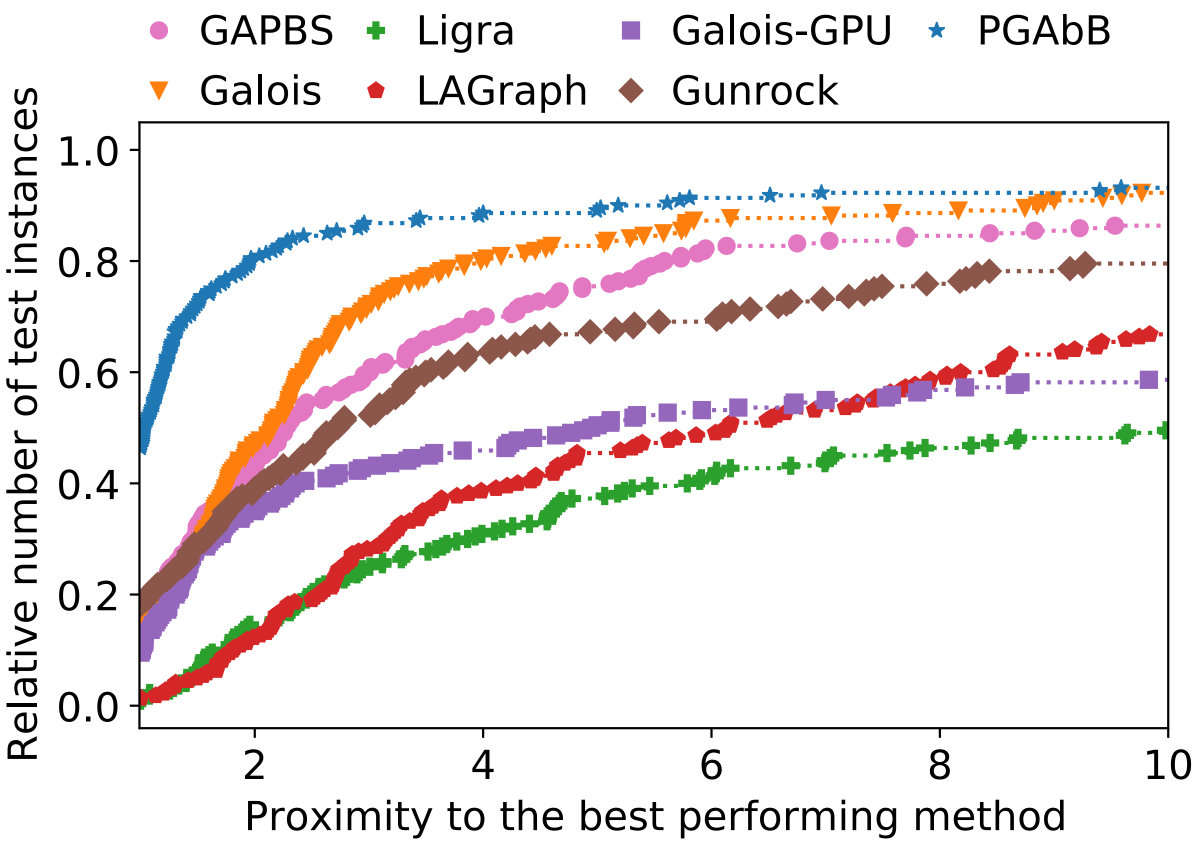
Selected Dataset

Graph	Number of Vertices	Number of Edges	Number of Triangles	Clustering Coefficient
Twitter7	41.6 M	1.2 B	34.8 B	0.001
Com-Orkut	3 M	117 M	627 M	0.041
Sk-2005	50.6 M	1.8 B	84.9 B	0.002
Kmer_V1r	214 M	232 M	49	0.000
Europe-OSM	50.9 M	54.1 M	61 K	0.003
Myciel.19	393 K	451 M	0	0
Kron-Scale21	2.1 M	91 M	8.8 B	0.044

Experiments on Selected Graphs

		Social		Web	Gene	Road	Synthetic	
		twitter7	Orkut	sk-2005	kmer_V1r	eu_osm	myciel19	kron21
Galois	PR	0.83	1.01	1.01	0.89	1.03	6.96	0.78
	SV/LP	8.40	1.71	1.68	2.29	1.81	1.25	1.12
	CC	0.84	1.56	0.98	0.64	0.64	2.94	0.81
	BFS	0.26	0.59	0.46	0.34	2.14	0.39	0.18
	TC	0.69	1.06	0.63	0.90	1.21	0.44	0.40
Ligra	PR	0.39	0.60	0.99	0.43	0.53	2.59	0.72
	SV/LP	1.24	0.70	1.05	0.18	0.02	0.58	0.66
	CC	0.02	0.04	0.00	0.02	0.01	0.03	0.02
	BFS	0.61	0.67	0.93	0.68	0.16	1.37	0.82
	TC	0.31	0.35	0.12	0.30	0.17	0.43	0.69
LAGraph	PR	0.75	0.98	0.60	0.75	0.65	3.21	0.71
	SV/LP	14.24	1.64	0.89	0.30	0.13	7.70	0.92
	CC	0.17	0.21	0.12	0.14	0.05	0.27	0.09
	BFS	0.79	0.33	0.77	0.27	0.33	0.75	0.30
	TC	0.38	0.87	0.66	0.29	0.16	0.52	0.37
Galois-GPU	PR	0.00	2.72	0.00	1.01	1.49	12.12	1.62
	SV/LP	0.00	3.67	0.00	2.43	2.71	2.65	1.57
	CC	0.00	0.46	0.00	1.16	0.99	0.09	0.15
	BFS	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	TC	1.03	0.85	0.90	0.00	0.00	0.38	0.65
Gunrock	PR	0.00	1.28	0.00	1.44	1.34	5.42	0.97
	SV/LP	0.00	1.88	0.00	3.18	1.22	3.90	0.97
	CC	0.00	0.24	0.00	1.51	0.44	0.14	0.09
	BFS	4.61	1.48	0.00	3.59	0.80	3.45	5.73
	TC	0.00	0.74	0.00	0.04	0.02	0.29	0.23
PGAbB	PR	4.64	4.67	0.80	0.53	0.64	10.76	1.79
	SV/LP	18.02	5.95	1.90	5.73	2.95	7.70	1.98
	CC	1.25	1.53	2.14	1.91	0.96	2.40	0.87
	BFS	0.16	0.89	0.77	0.90	0.33	1.00	0.29
	TC	3.02	3.01	1.69	1.11	3.91	5.39	3.48

Overall Comparison



PGAbB performs 1.6x to 5.7x better than state-of-the-art in the median.

Galois performs the second. GAPBS performs the third.

Conclusion and Future Work

In this work we proposed PGAbB which provides

- an easy block-based programming model for leveraging heterogenous architectures.
- computation and data partitioning strategies for maximal usage of the available resources.
- simple and effective scheduling strategies for CPU and/or GPU processing of different graph kernels.

We are currently working on:

- Simplifying the user API.
- Memory hierarchy aware smarter block fetching.
- Open-source software release.
- Future work: Hypergraph-based locality aware different scheduling policies.

TDAlab Members and Collaborators

Triangle Count / PGAbB


Abdurrahman Yaşar


Sivasankaran Rajamanickam


Jonathan Berry

Current TDAlab Members

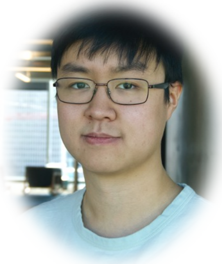

Ümit V. Çatalyürek


Abdurrahman Yaşar


Yusuf Özkaya


Kasimir Gabert


Xiaojing An


James Fox


Kaan Sancak


Fatih Balin

Benjamin Cobb

Thanks

- For more information
  - email umit@gatech.edu
  - Visit tda.gatech.edu
- Acknowledgement of Support