miniVite: A Graph Analytics Benchmarking Tool for Massively Parallel Systems

Sayan Ghosh[*], *Mahantesh Halappanavar*[Ψ], Antonino Tumeo[Ψ], Ananth Kalyanaraman[*], Assefaw Gebremedhin[*]

[*]Washington State University, Pullman, WA

[Ψ]Pacific Northwest National Laboratory, Richland, WA

2018 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)

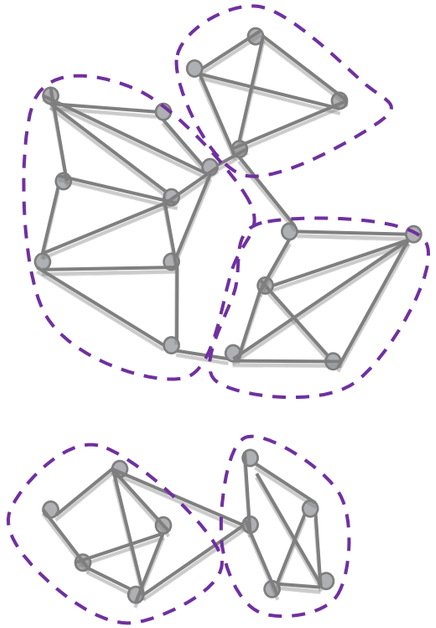
12 Nov, 2018. Dallas, Texas.

Graph Clustering (community detection)

- Problem: Given $G(V,E,\omega)$, identify tightly knit groups of vertices that strongly correlate to one another within their group, and sparsely so, outside.

Input :

- $V = \{1,2,\ldots n\}$
- E: a set of M edges
- $\omega(e)$: weight of edge e (non-negative)
- $m = \sum_{\forall e \in E} \omega(e)$

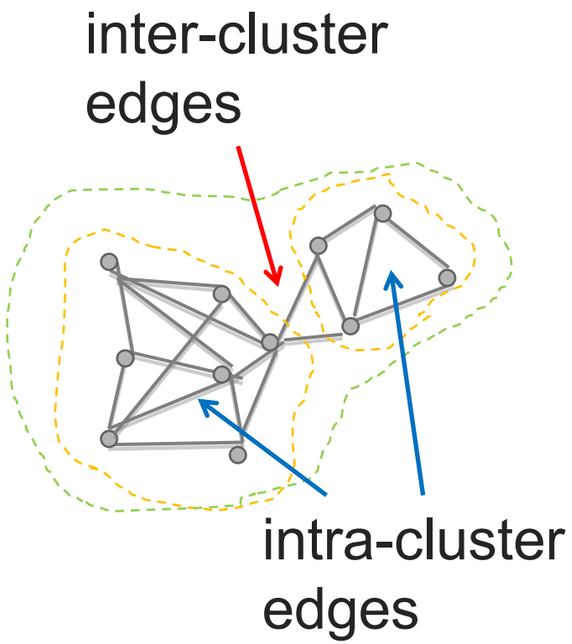


Output :

- A partitioning of V into k mutually disjoint clusters $P = \{C_1, C_2, \ldots C_k\}$ such that: … ?

Modularity (Newman 2004)

- A statistical measure for assessing the quality of a given community-wise partitioning *P* of the vertices *V*:

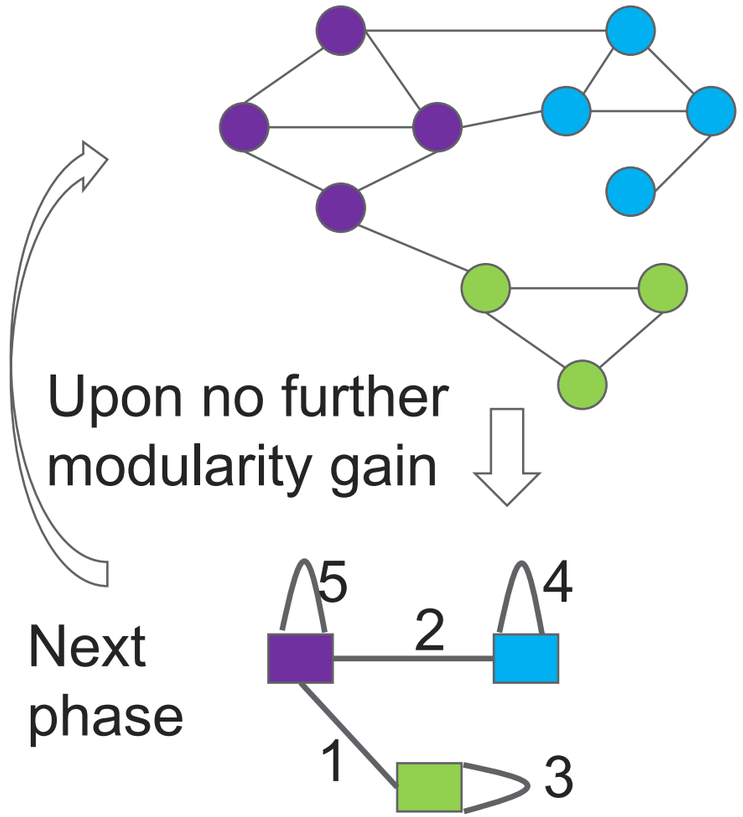


Notation	Definition
$C(i)$	Cluster containing vertex i
$e_{i->C(i)}$	Number of edges from *i* to vertices in $C(i)$
a_C	Sum of the degree of all vertices in cluster *C*

$$Q = \underbrace{\frac{1}{2m}\sum_{\forall i \in V} e_{i \to C(i)}}_{\text{Fraction of intra-cluster edges}} - \underbrace{\sum_{\forall C \in P} \left(\frac{a_C}{2m} \cdot \frac{a_C}{2m}\right)}_{\text{Equivalent fraction in a random graph}}$$

Louvain method (Blondel *et al*. 2008)

Input: G(V,E)
Goal: Compute a partitioning of V that maximizes modularity (Q)
Init: Every vertex starts in its own community (i.e., *C(i)={i}*)

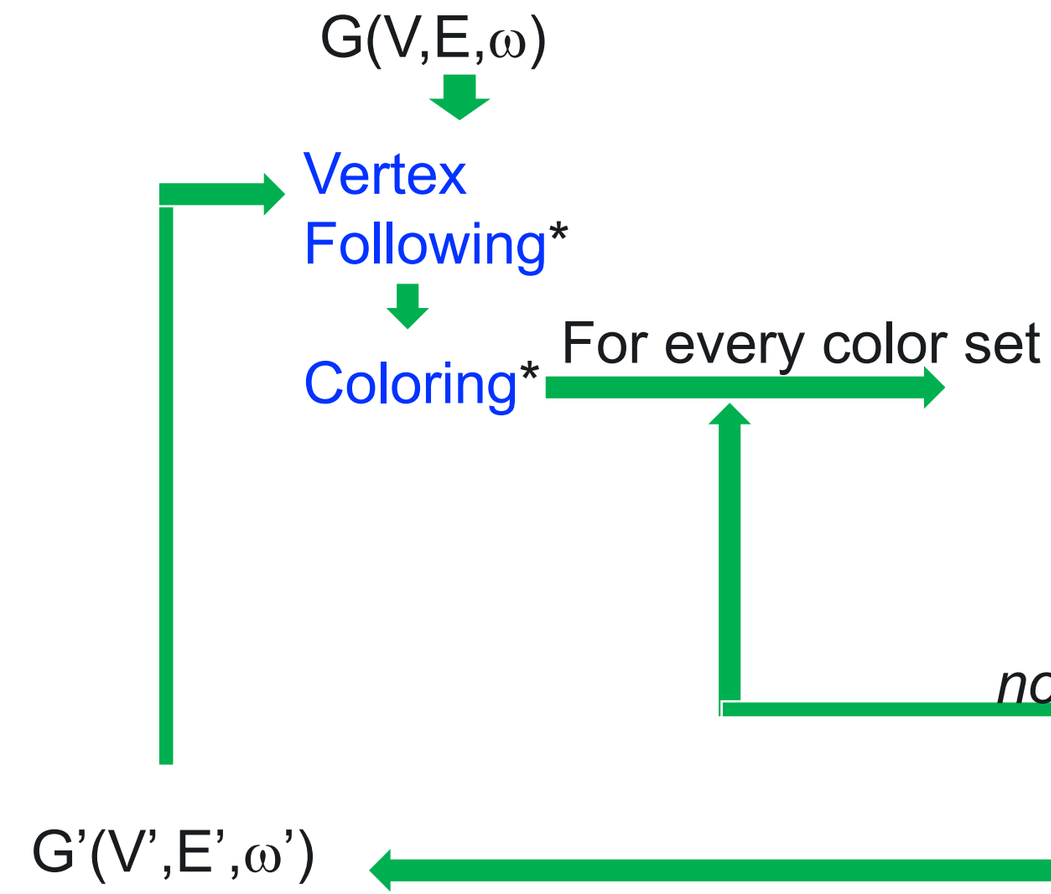


Multi-phase multi-iterative heuristic

Within each iteration:

- **For** every vertex $i \in V$:
  1. Let *C(i)* : current community of *i*
  2. Compute modularity gain (ΔQ) for moving *i* into each of *i*'s neighboring communities
  3. Let C_{max} : neighboring community with largest ΔQ
  4. If ($\Delta Q > 0$) { Set $C(i) = C_{max}$ }

Our Parallel Algorithm: *Grappolo*



PARALLEL FOR each vertex $i \in V_c$:

1. Let *C(i)* : current community of *i*
2. Compute modularity gain (ΔQ) for moving *i* into each of *i*'s neighboring communities
3. Let C_{max} : neighboring community with largest ΔQ *(ML)*
4. If ($\Delta Q > 0$) { Set *C(i)* = C_{max} }

* Steps are optional

Rebuilding is nontrivial, but takes 1-10% of total time

Distributed Grappolo: Vite

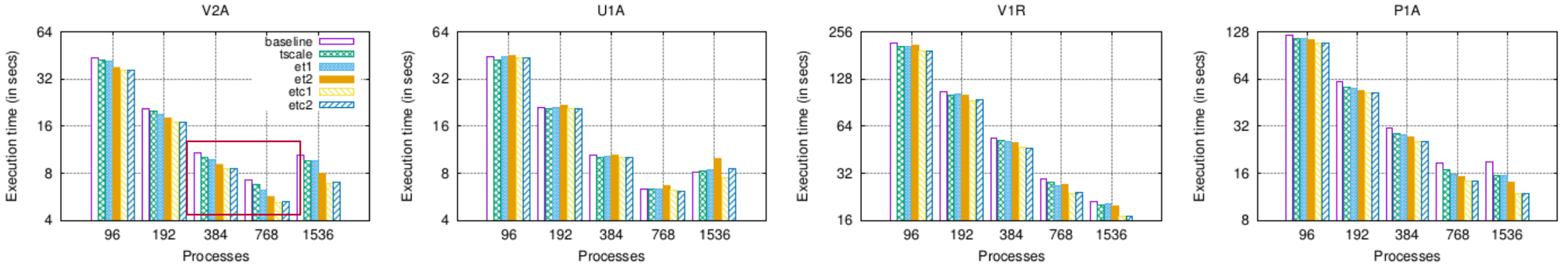


Fig. 7. Scalability of protein k-mer graphs.

We implement heuristics on top of the baseline distributed version, yielding speedups of up to **2.5-46x** (compared to baseline), modularity affects sometimes by ~8-20%

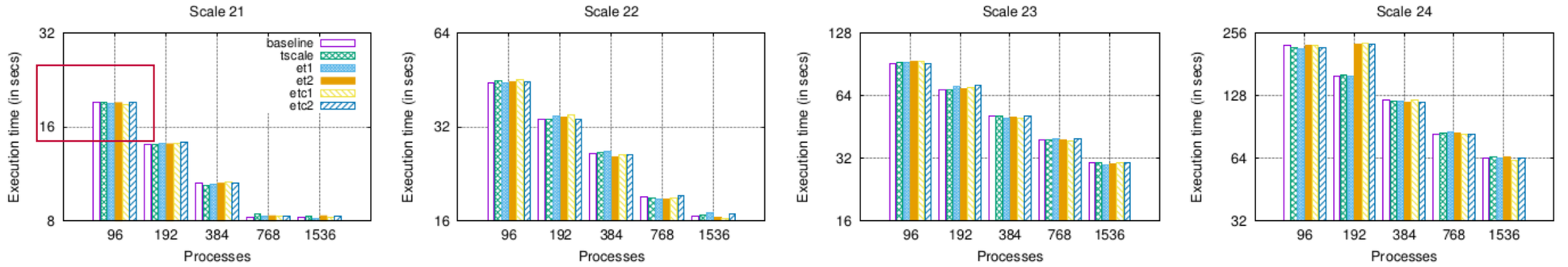


Fig. 1. Parallel heuristics have little effect on RMAT generated Graph500 graphs.

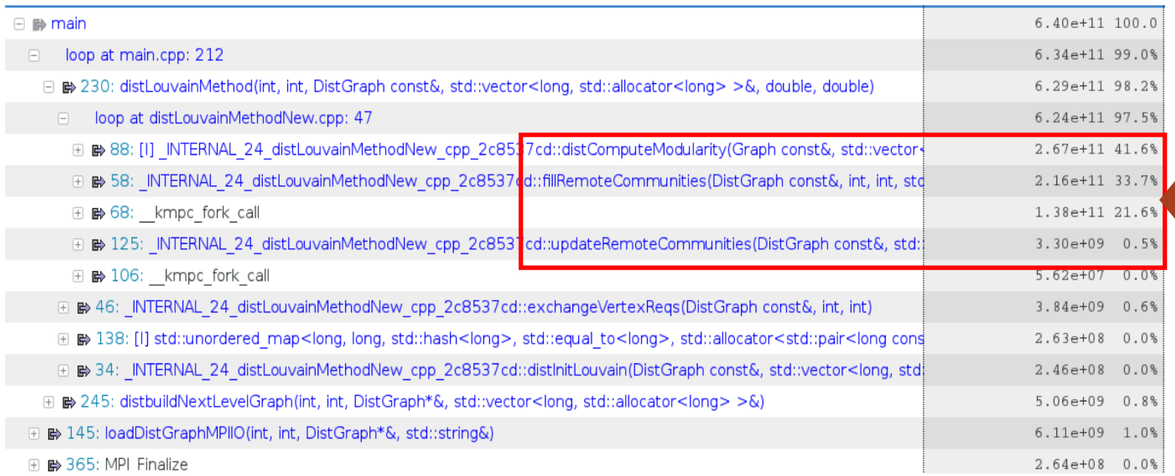
However, heuristics have little impact for some inputs!

Our goal: To study the baseline version: Communication options, data structures, etc.

Observations from Vite

Graphs	#Vertices	#Edges	First phase			Complete execution			
			Iterations	Modularity	Time	Phases	Iterations	Modularity	Time
friendster	65.6M	1.8B	143	0.619	565.201	3	440	0.624	567.173
it-2004	41.3M	1.15B	14	0.394	45.064	4	91	0.973	45.849
nlpkkt240	27.9M	401.2M	3	0.143	3.57	5	832	0.939	21.084
sk-2005	50.6M	1.9B	11	0.314	71.562	4	83	0.971	72.94
orkut	3M	117.1M	89	0.643	59.5	3	281	0.658	59.64
sinaweibo	58.6M	261.3M	3	0.198	270.254	4	108	0.482	281.216
twitter-2010	21.2M	265M	3	0.028	209.385	4	184	0.478	386.483
uk2007	105.8M	3.3B	9	0.431	35.174	6	139	0.972	37.988
web-cc12-paylvladmin	42.8M	1.2B	31	0.541	140.493	4	159	0.687	146.92
webbase-2001	118M	1B	14	0.458	14.702	7	239	0.983	24.455

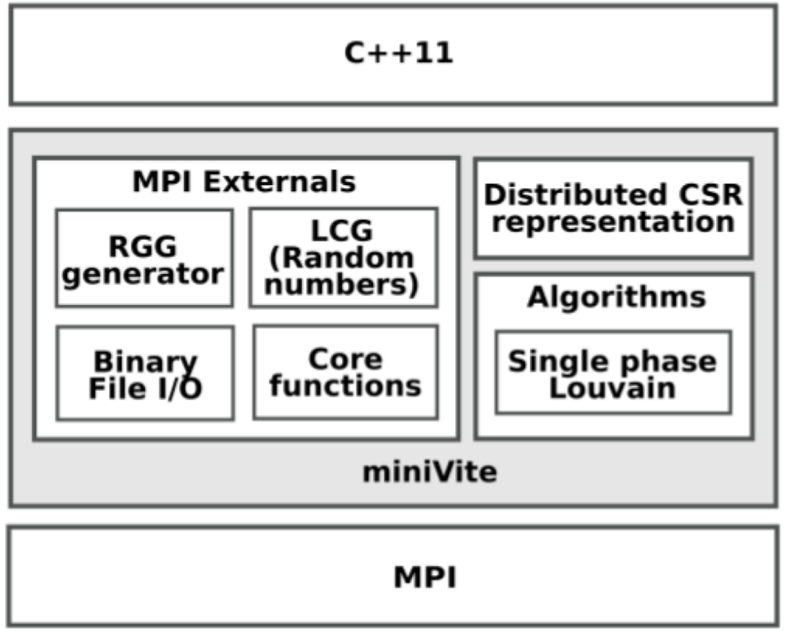
For a number of real world graphs, the first phase of Louvain method does most of the work (little difference between first and final phase)

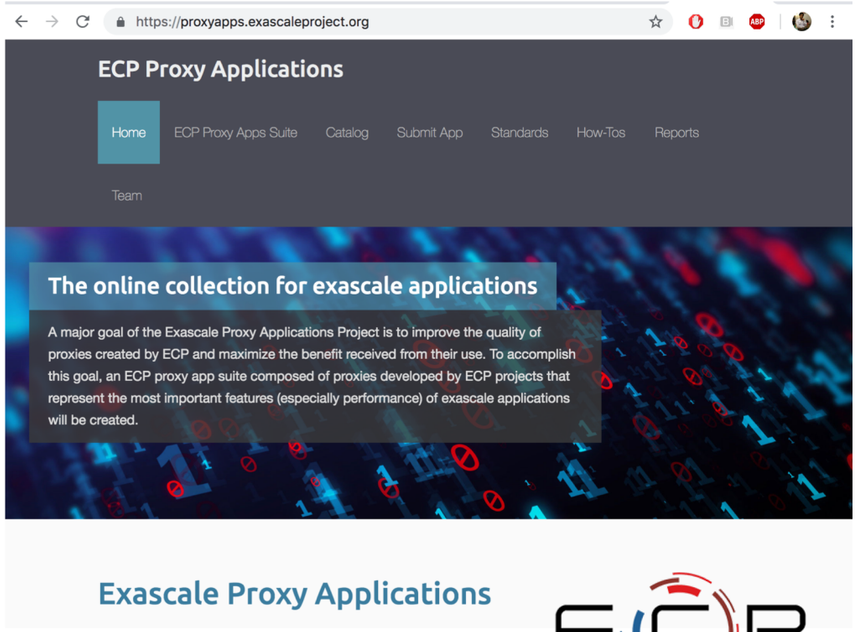


1. HPCToolkit profiling shows over **60%** of time is spent in managing and communicating vertex-community information
2. About **40%** is spent on global communication (MPI_Allreduce) for computing modularity

miniVite (/'vi:te/)

- Implements a single phase of Louvain method (without rebuilding the graph)
- Capable of generating synthetic Random Geometric Graphs (RGG) in parallel (needs random numbers)
  - Can also add random edges across processes
- Can also use real world graphs as input (have to convert to a binary format first)
- Parts of code has multiple communication options (can be selected at compile time) – Sendrecv, NB Isend/Irecv (default), MPI RMA and Collectives
- About 3K LoC

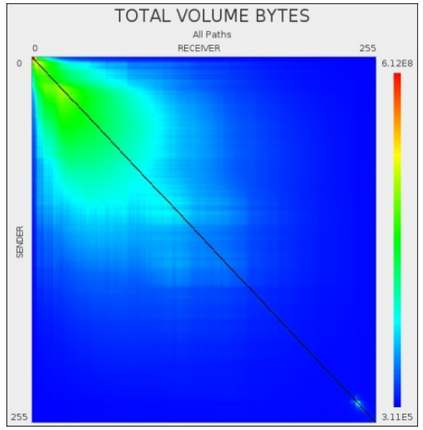




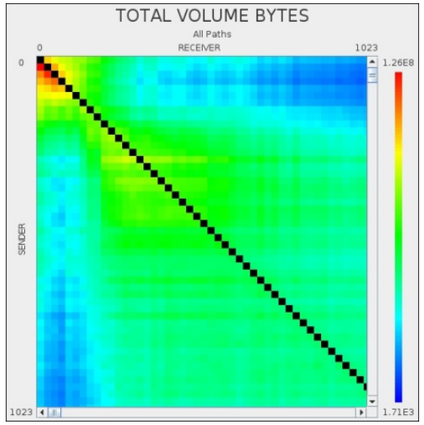
Pacific Northwest NATIONAL LABORATORY

Why clustering?

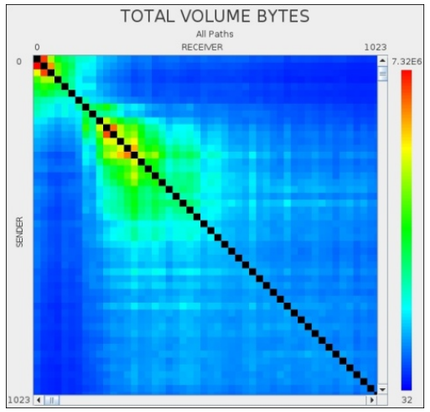
- Computation: Performs some computation (modularity), whereas other graph workloads may have 0 FLOPS
- Communication intensive: In every iteration, as a vertex migrates, {size, degree} of communities change and ghost communities have to exchange information accordingly
- Nondeterministic: Execution time is sensitive to structure and sizes of input (#iterations, #clusters, relative sizes)
- Dynamic: Process neighborhood changes in every phase, as graph gets rebuilt



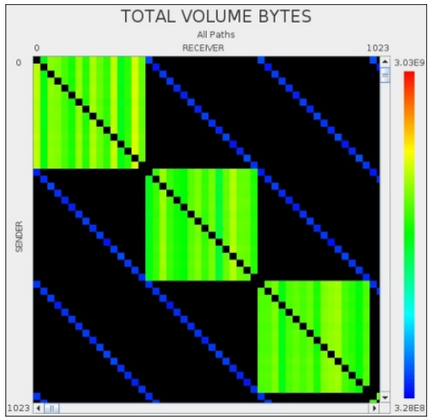
Louvain method on 256 PEs, Friendster (1.8B edges)



Louvain method on 1K PEs, Friendster (1.8B edges)



1/2-approx matching on 1K PEs, Friendster (1.8B edges)

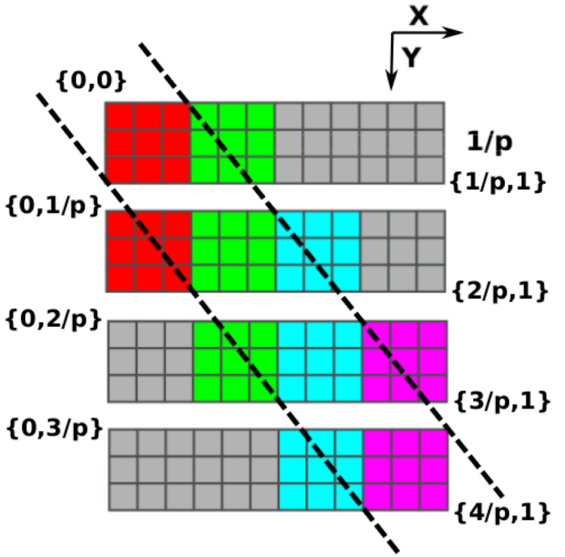


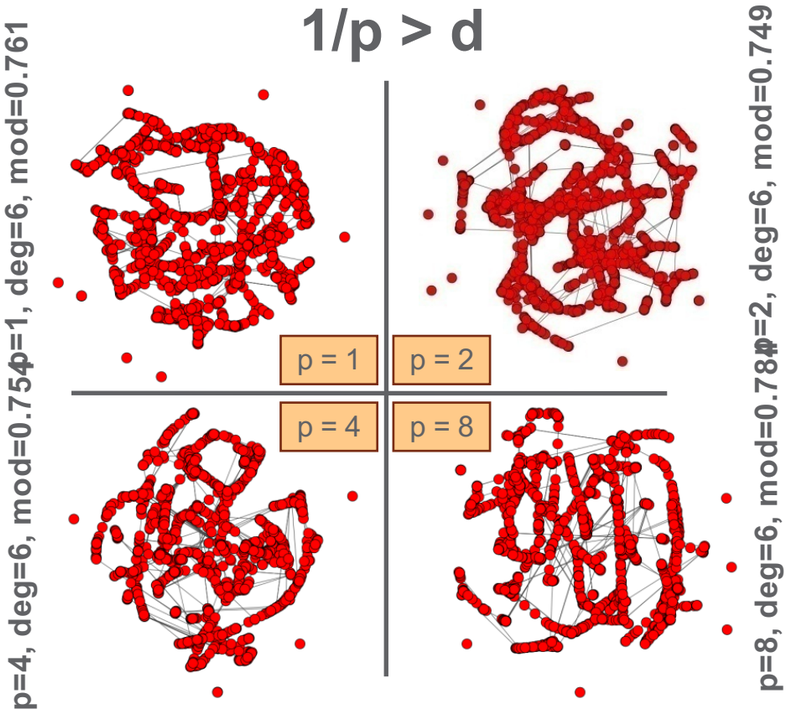
Graph500 BFS on 1K PEs, Scale 27 (2.14B edges)

Communication heat maps

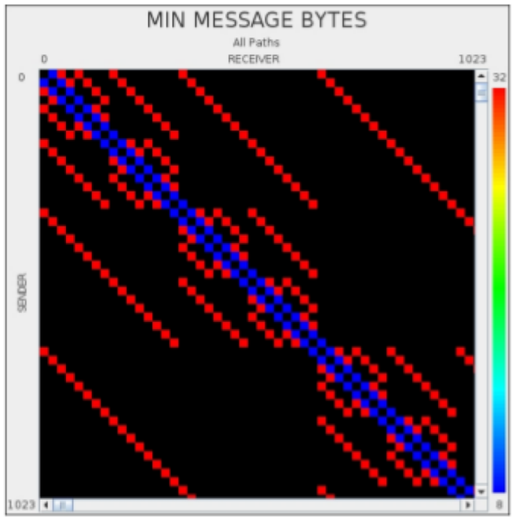
Random Geometric Graph

- Constructed by randomly placing *N* nodes within a unit square – only add an edge between two vertices if their distance is within a range *d*
  - RGG is known to demonstrate good community structure (meaningful)
- We distribute equal number of vertices across processes, each process may have (cross) edges with its up and/or down neighbors
- Option to add random number of (cross) edges across processes that are farther apart

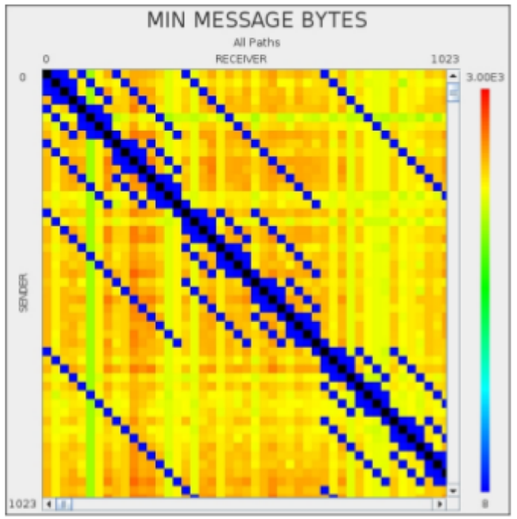




Communication characteristics

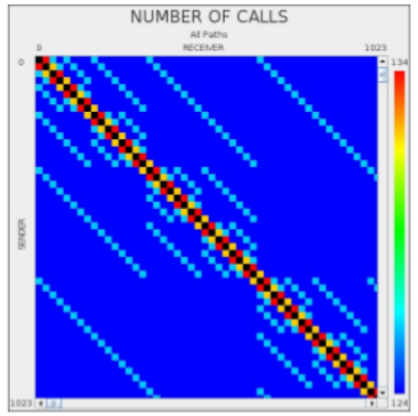


(a) Basic RGG input, black spot means zero exchange.

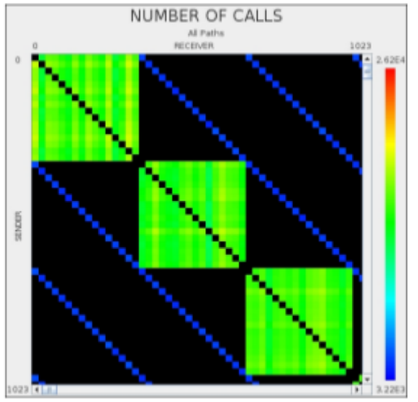


(b) RGG input with 20% extra edges.

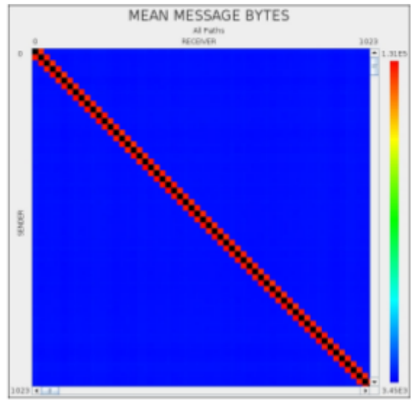
Process only communicates with 2 neighbors for basic RGG, however, communication patterns change when extra edges are added randomly (modularity changes as well)



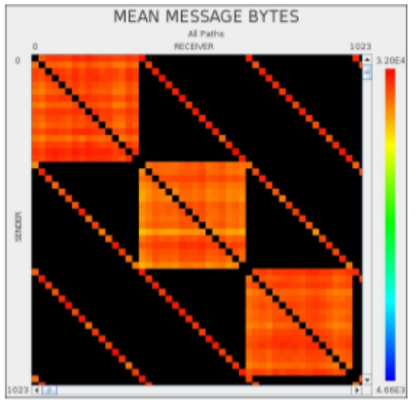
(a) MPI calls — `miniVite`.



(b) MPI calls — Graph500 BFS.



(c) Mean message sizes — `miniVite`.



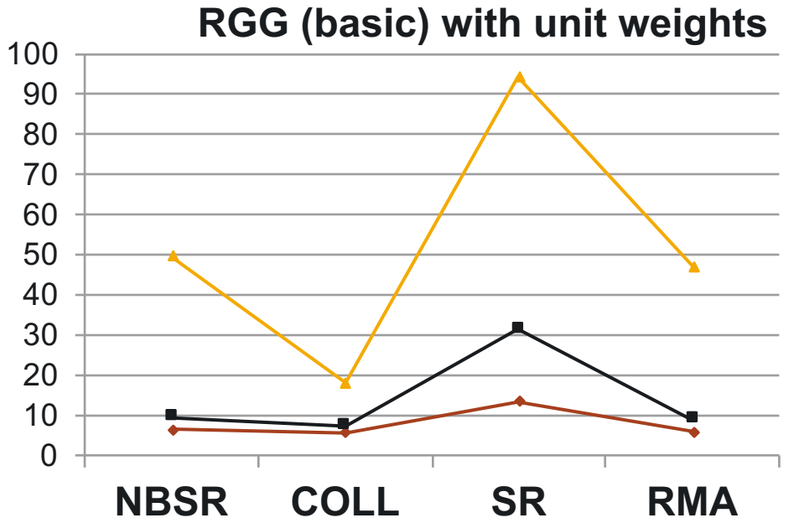
(d) Mean message sizes — Graph500 BFS.

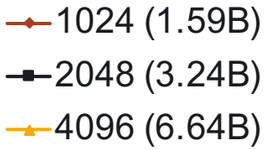
Clustering

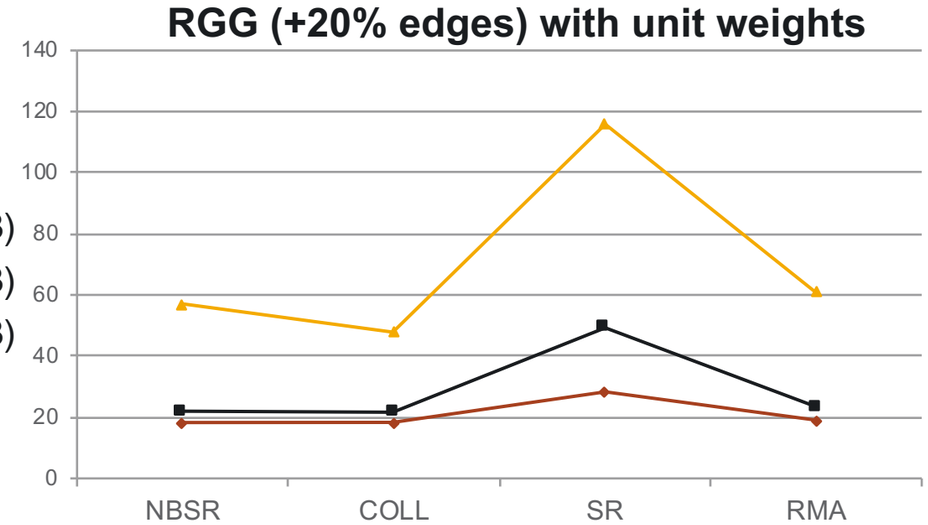
BFS

No dark spots, reasonable communication volume per process

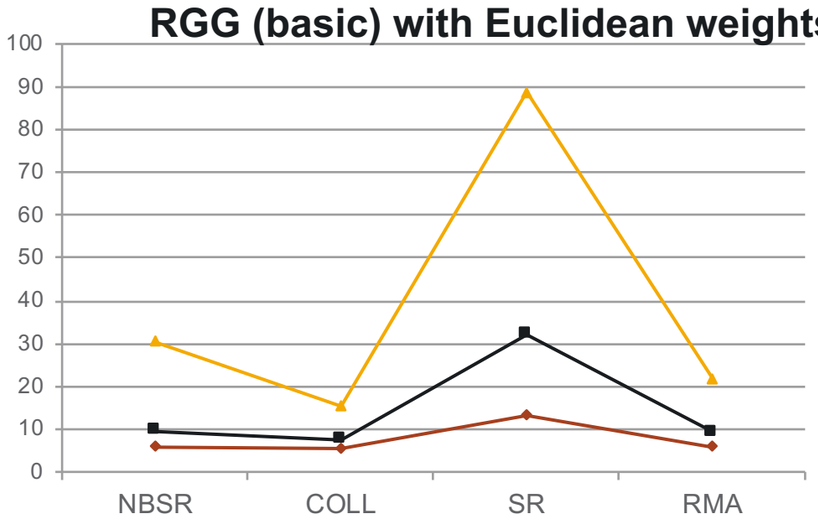
Impact of communication models

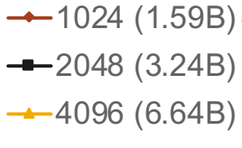


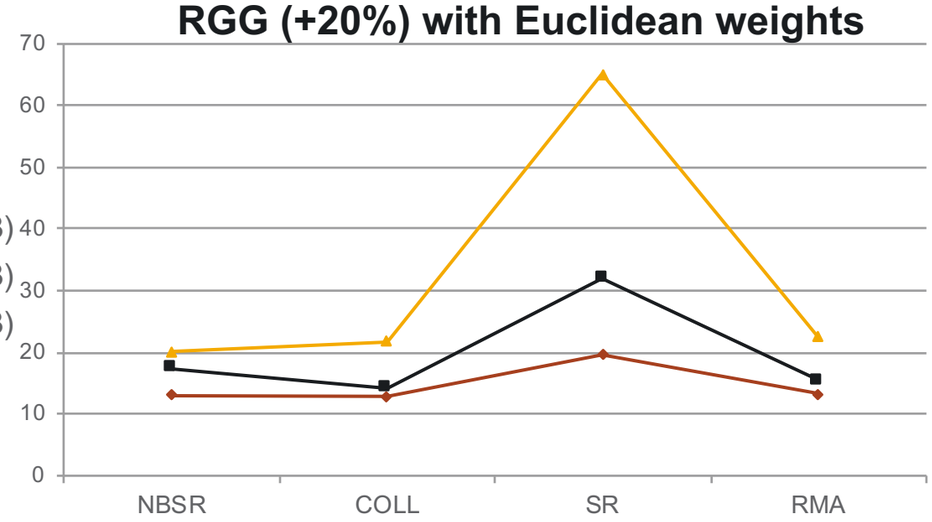




Weak scaling runs



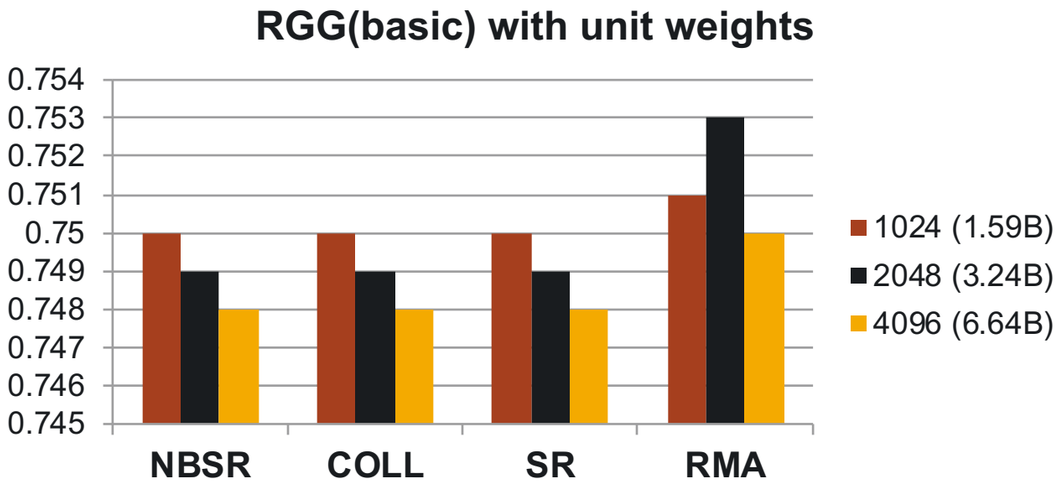


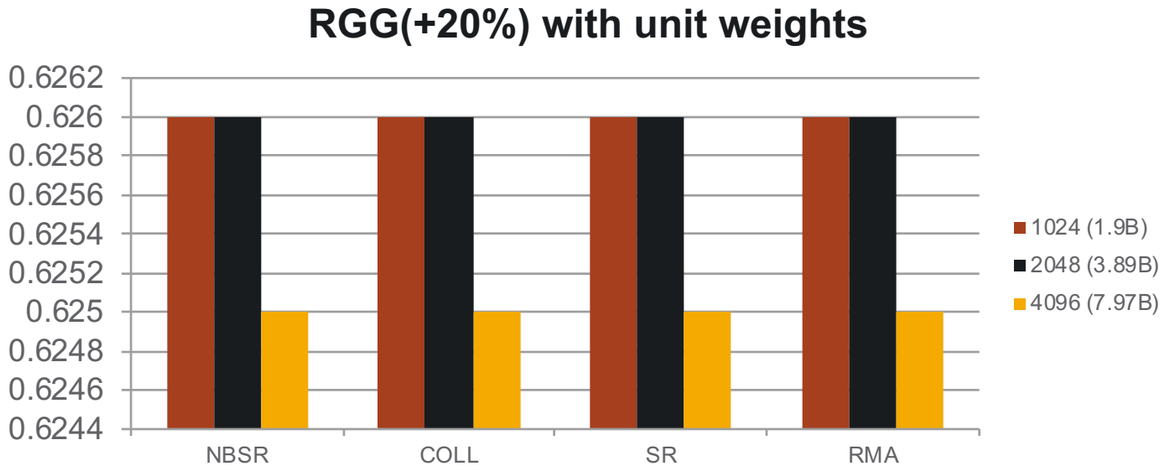


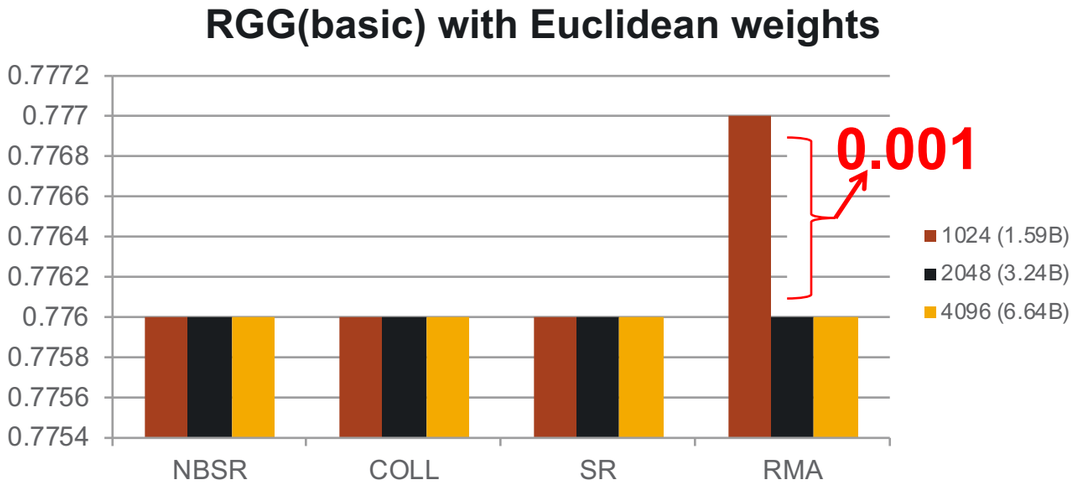
Subpar MPI_Sendrecv implementation?

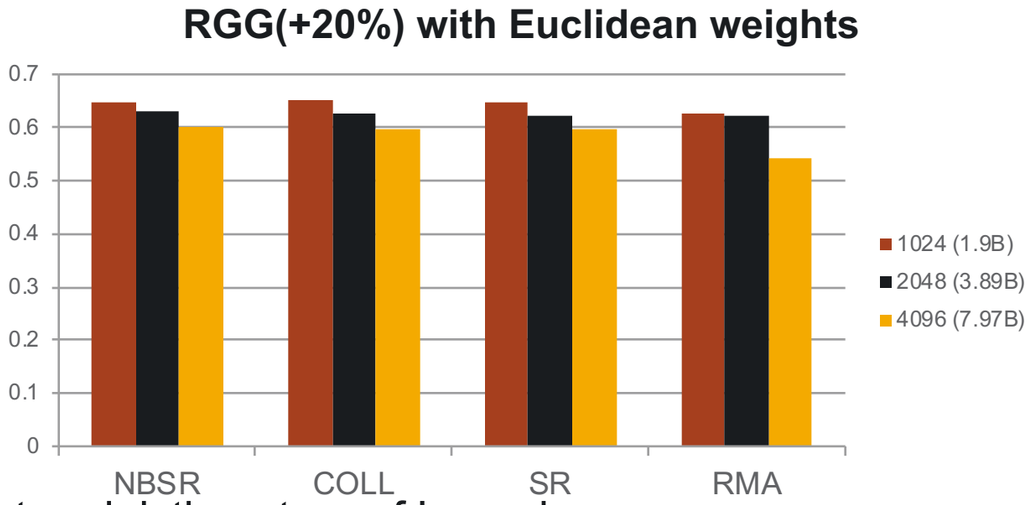
Execution time increases with extra edges (up to 3x)

Impact on modularity









Minor variation in modularity, owing to non-deterministic nature of Louvain

Strong scaling (Friendster, 1.8B edges)

Number of iterations, execution time (in secs.) and Modularity (Q) of Friendster (65.6M vertices, 1.8B edges) on 1024/2048 processes.

Versions	1024 processes			2048 processes		
	Itrs	Time	Q	Itrs	Time	Q
NBSR	111	745.80	0.6155	127	498.89	0.6177
COLL	109	752.41	0.6159	141	554.98	0.6204
SR	111	783.94	0.6157	103	423.43	0.6191
RMA	109	782.47	0.6162	111	589.47	0.6190

Dissimilar number of iterations across versions affect execution time and modularities

Pacific Northwest NATIONAL LABORATORY

Concluding remarks

- **miniVite** serves as a sandbox for assessing performance of different communication primitives, quality of heuristics, correctness, and understanding impact of different datasets
- Can generate different datasets due to in-memory graph generator, extra options that may impact communication
- Code released as part of ECP Proxy Apps suite

https://proxyapps.exascaleproject.org/app/minivite

Pacific Northwest
NATIONAL LABORATORY

hala@pnnl.gov

Thank you

- DOE Exascale Computing Project
- Dave Richards and Abhinav Bhatele (LLNL) – Thanks for the push!

git clone https://github.com/Exa-Graph/miniVite.git