Proxy application for Graph Community Detection or Clustering

Sayan Ghosh, Mahantesh Halappanavar, Antonino Tumeo, Nathan Tallent

Pacific Northwest National Laboratory, Richland, WA

2022 IndySCC competition webinar
10/3/2022

Graph Clustering or Community Detection

- Problem: Given $G(V,E,\omega)$, identify tightly knit groups that strongly correlate to one another within their group, and sparsely so, outside.
  - Output: A partitioning of V into k mutually disjoint clusters $P = \{C_1, C_2, \ldots C_k\}$ such that: … ?
  - Modularity (Newman, 2004) is a metric for assessing quality of P

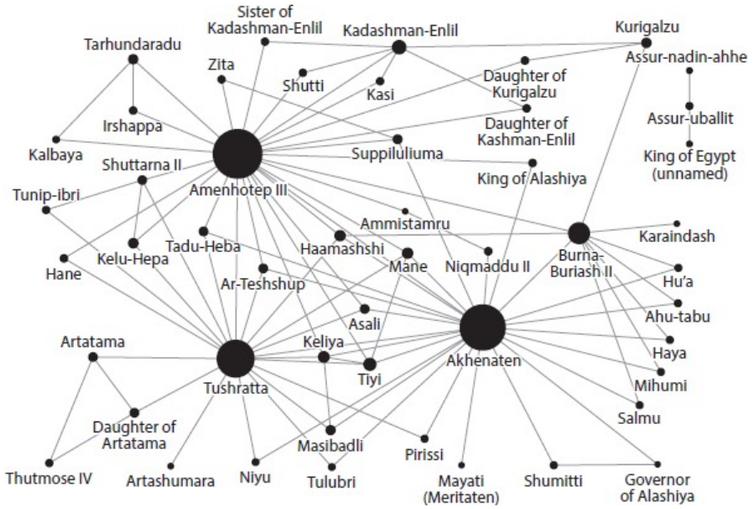
Notation	Definition
$C(i)$	Cluster containing vertex i
$e^{i \to C(i)}$	Number of edges from i to vertices in $C(i)$
a_C	Sum of the degree of all vertices in cluster C

$$Q = \frac{1}{2m} \sum_{\forall i \in V} e_{i \to C(i)} - \sum_{\forall C \in P} \left(\frac{a_C}{2m} \cdot \frac{a_C}{2m} \right)$$

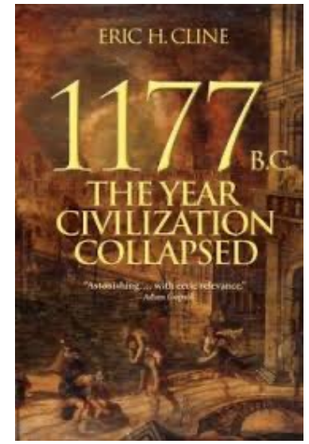
Sum of all edge-weights → m

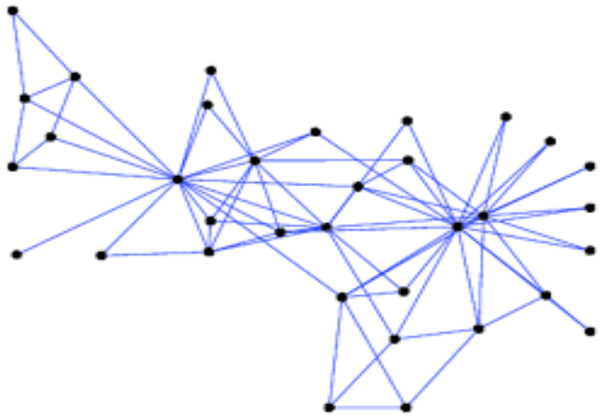
Fraction of intra-cluster edges

Equivalent fraction in a random graph



Amarna letters, c. 1360-32 BC

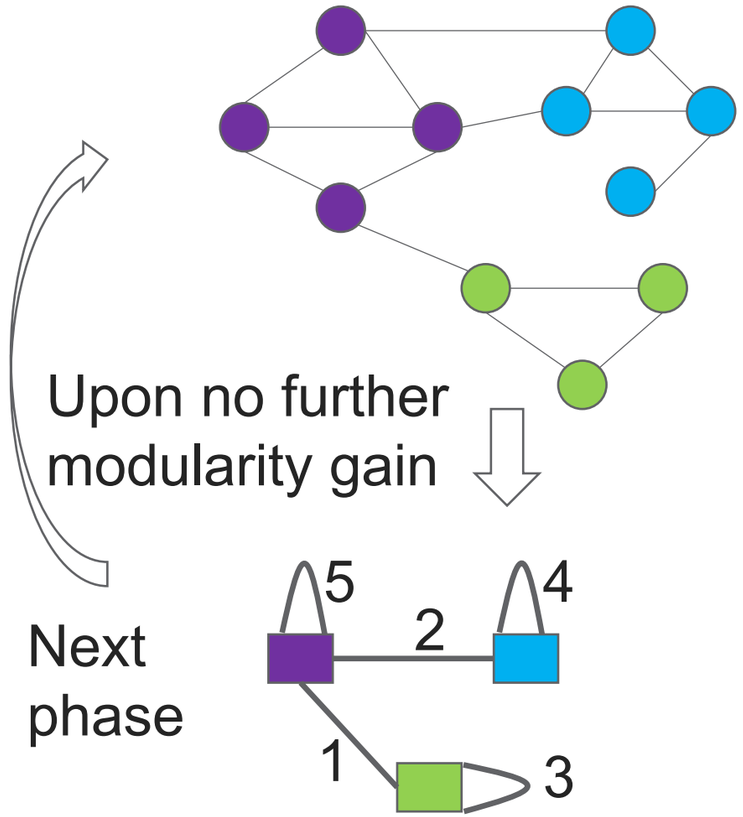




Zachary's Karate Club, c. 1970 AD

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 $C(i) = C_{max}$ }

Louvain method (Blondel *et al*. 2008)

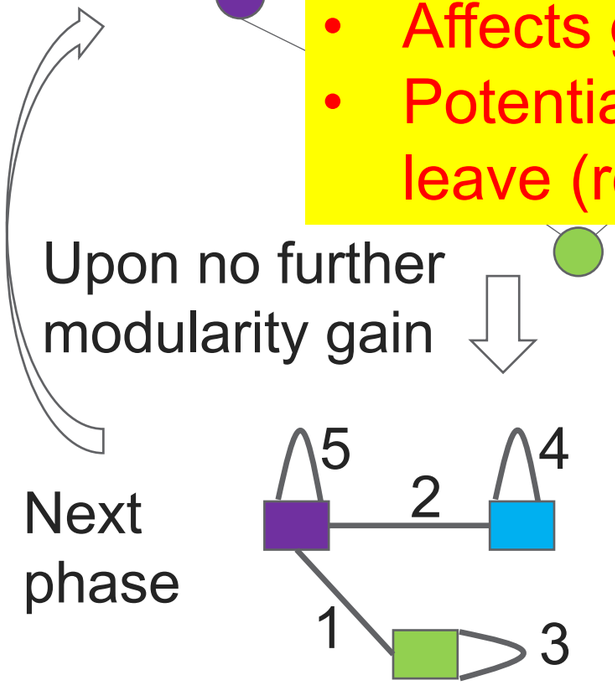
Input: G(V,E)

Goal: Compute a partitioning of V that maximizes modularity (Q)

Init: Every vertex starts in its own community (i.e., *C(i)={i}*)

Several parallelization challenges appear due to relaxed synchronization:

- Affects global modularity and convergence
- Potential load imbalance as vertices enter and leave (remote) communities



ristic

y 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 $C(i) = C_{max}$ }

Distributed Clustering: Vite

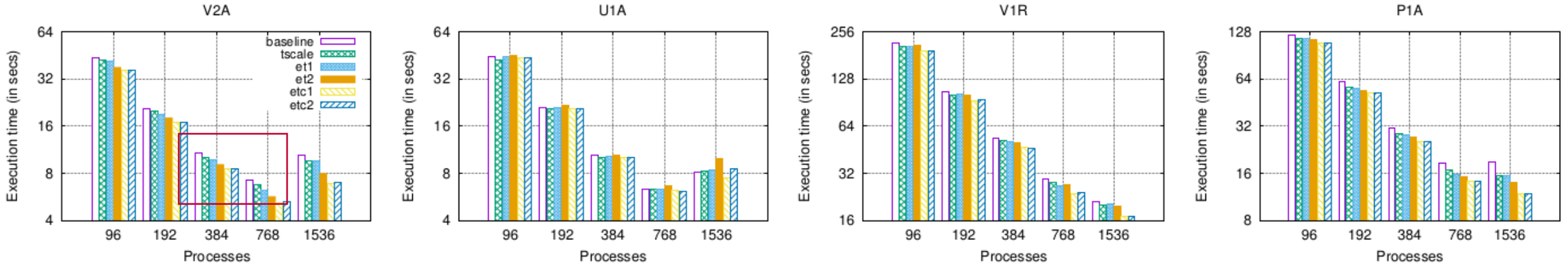


Fig. 7. Scalability of protein k-mer graphs.

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%

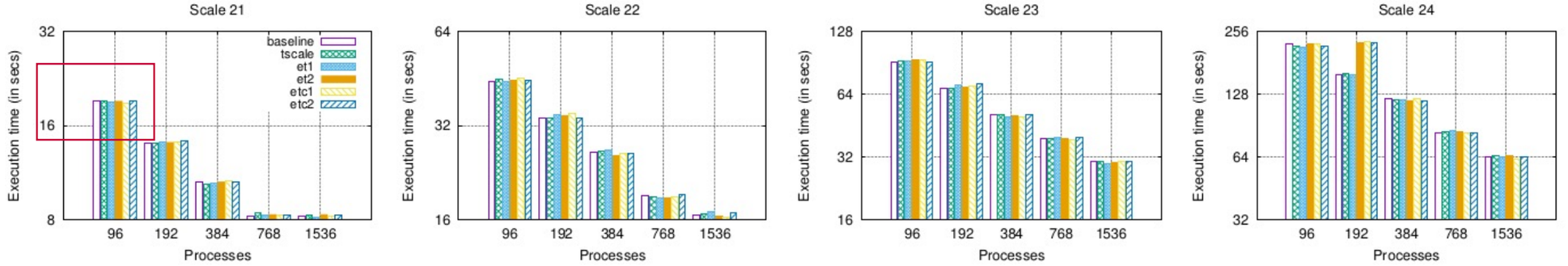


Fig. 1. Parallel heuristics have little effect on RMAT generated Graph500 graphs.

However, heuristics have little impact for some inputs!

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

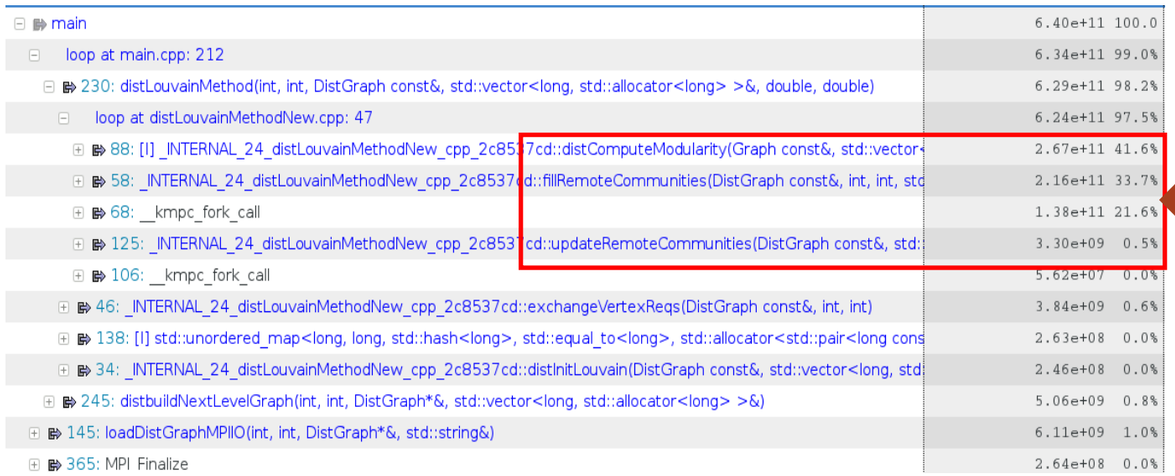
Observations from Vite

First phase execution time

Final execution time

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

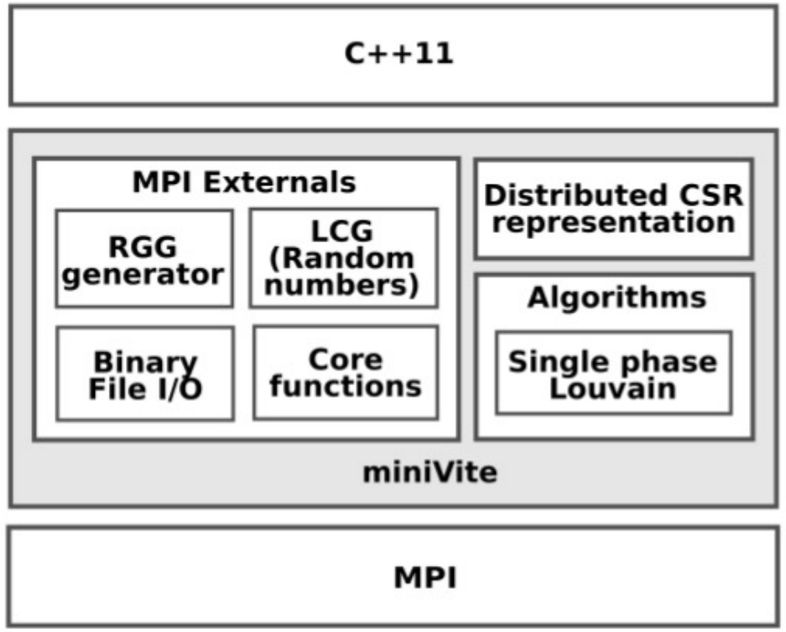
Proxy Application driven Codesign

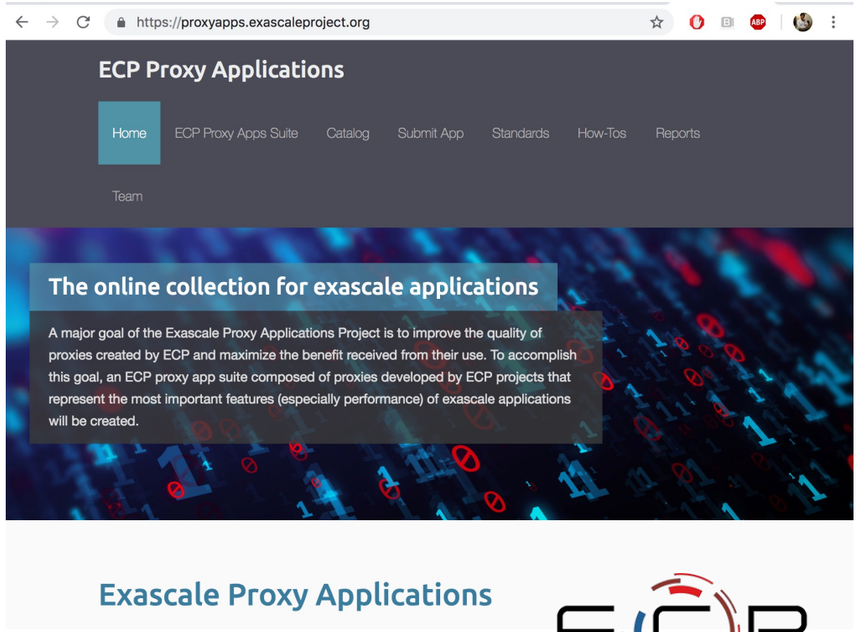
Coordinated design of applications, methods and architectures

- Applications are too complicated and rigid to be used as an analysis tool
- Proxy applications are representative of workloads that capture compute/throughput patterns in existing or prospective applications
  - Quantify bottlenecks in hardware and software – influence design of future systems
  - Expose trade-offs (e.g., memory and computation)
  - Sandbox to design and expand applications on future systems
  - Enable systematic profiling and analysis
  - Find limitations of underlying programming models and runtimes
  - Caters to a wide variety of researchers, vendors, tools/runtime/compiler developers, etc.

miniVite (/'vi:te/)

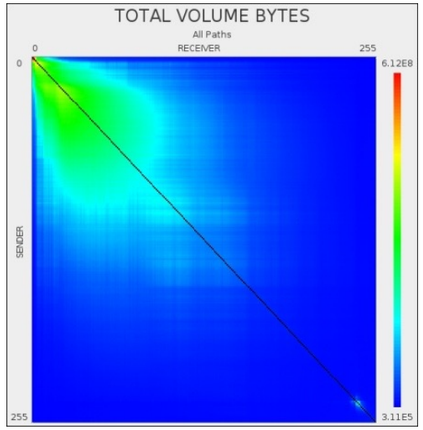
- Implements a single phase of Louvain method (without rebuilding the graph)
  - Similar computation/communication patterns to the parent application
- Capable of generating synthetic Random Geometric Graphs (RGG) in parallel (needs random numbers)
  - Can also add random edges across processes
- Can also use real world graphs as input (must 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



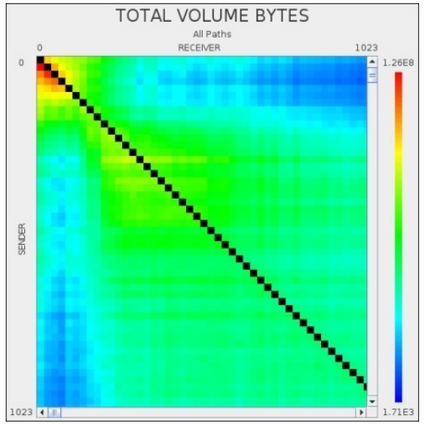


Why clustering?

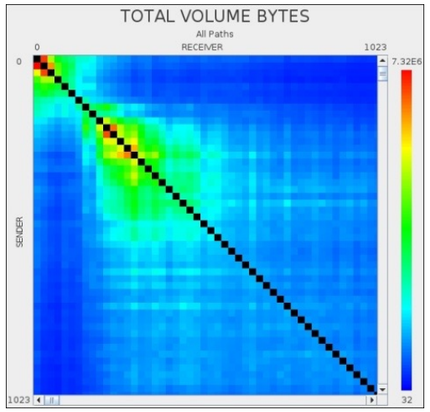
- Computation: Performs some computation (modularity), whereas other graph workloads may have 0 FLOPS
- Communication intensive: In every iteration, as a vertex migrates, {size, degree} of communities change and ghost communities have to exchange information accordingly
- Nondeterministic: Execution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



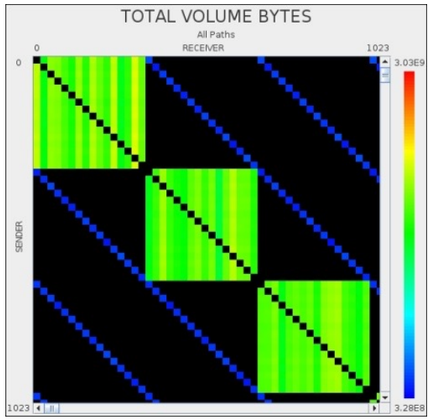
Louvain method on 256 PEs, Friendster (3.6B edges)



Louvain method on 1K PEs, Friendster (3.6B edges)



1/2-approx matching on 1K PEs, Friendster (3.6B edges)

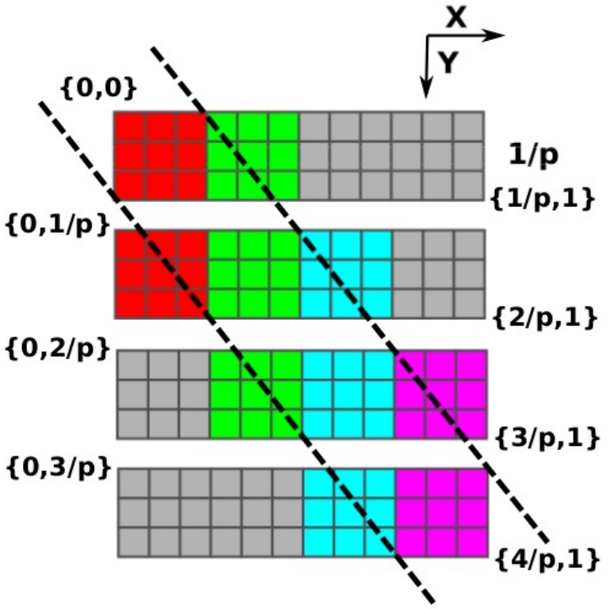


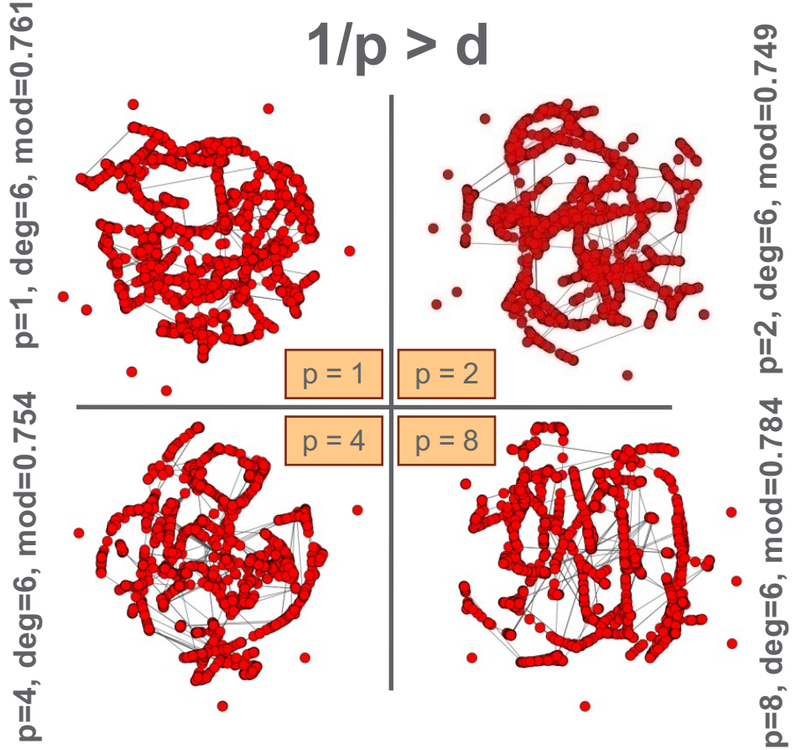
Graph500 BFS on 1K PEs, Scale 27 (2.14B edges)

Communication heat maps

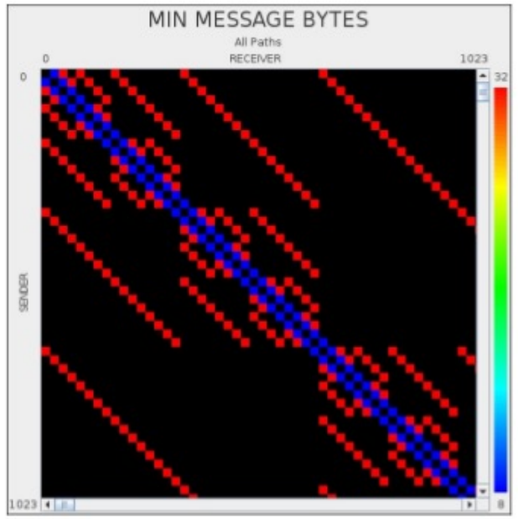
In-memory Random Geometric Graph generation

- Random Geometric Graphs: 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

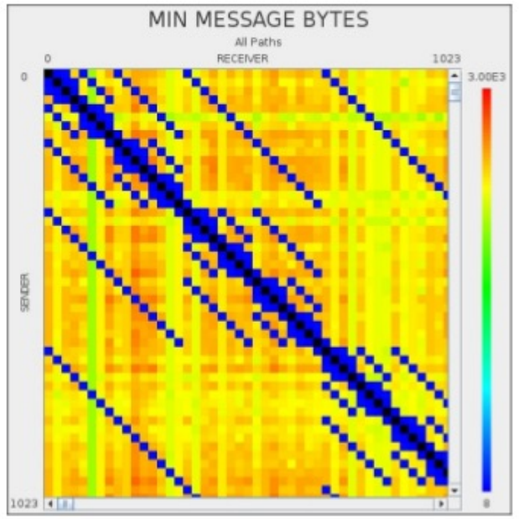




Communication characteristics

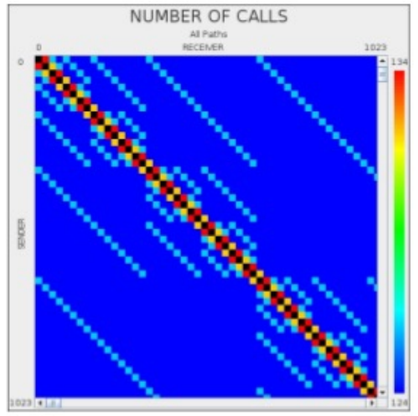


(a) Basic RGG input, black spot means zero exchange.

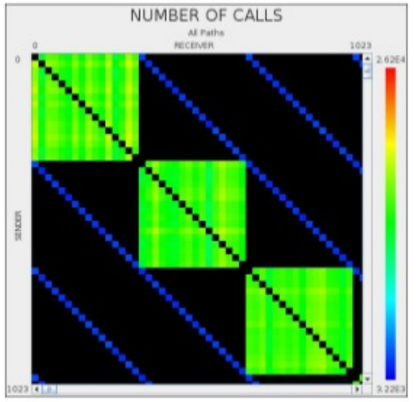


(b) RGG input with 20% extra edges.

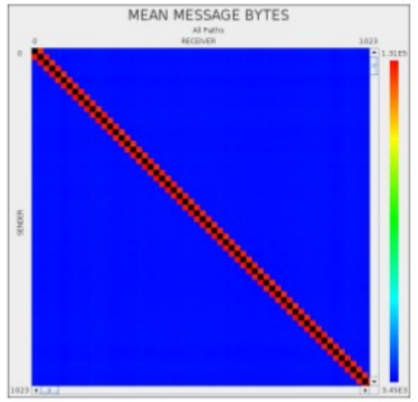
Process only communicates with 2 neighbors for basic RGG, however, communication patterns change when extra edges are added randomly (modularity changes as well)



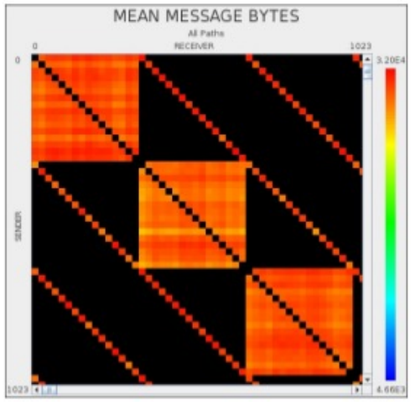
(a) MPI calls — `miniVite`.



(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

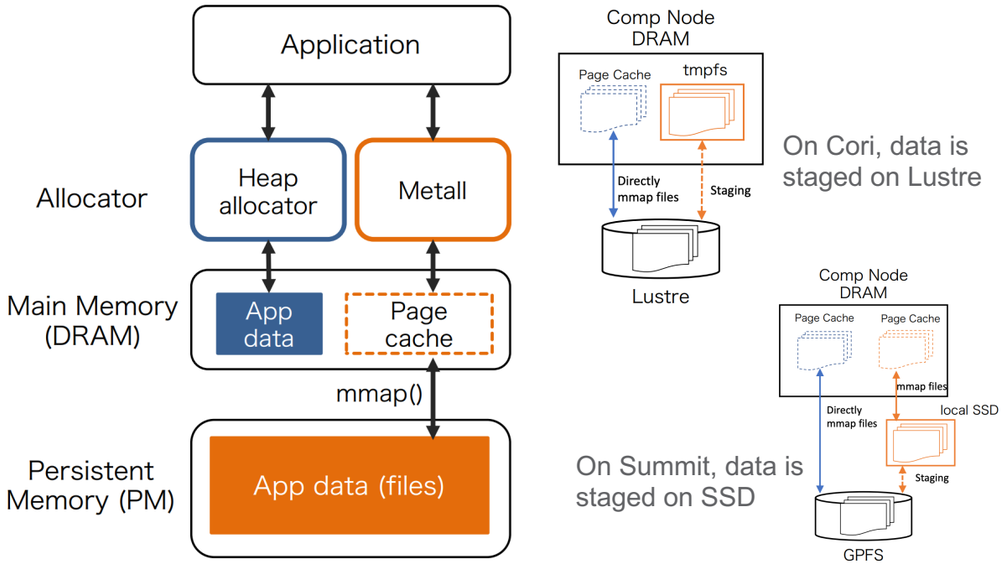
Number of iterations, execution time (in secs.) and Modularity (Q) of Friendster (65.6M vertices, 3.6B edges) on 1024/2048 processes.

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

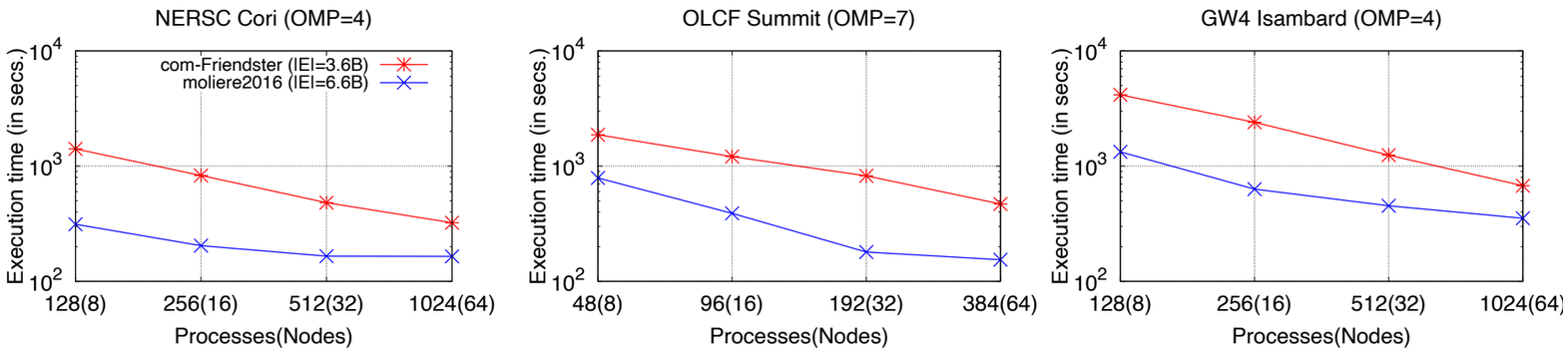
With increasing number of processes, dissimilar number of iterations across versions affect execution time

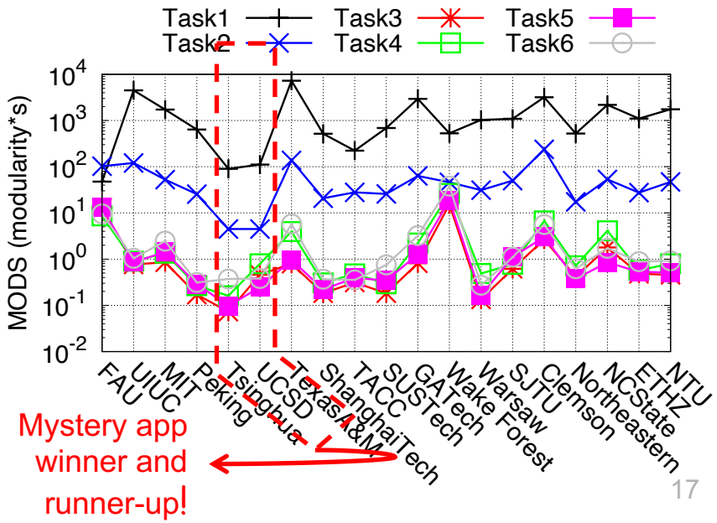
Related activities

- Used by vendors and researchers to assess and study performance of irregular workloads
- Preliminary checkpoint/restart
  - Supports storing/loading graph via LLNL metall allocator, negligible performance difference
- Checking suitability of MODS figure-of-merit
  - Assessed *mystery application* results in VSCC'20
- 30-50% difference at scale across platforms!



*Iwabuchi K, Ghosh S, Pearce R, Halappanavar M, Gokhale M. miniVite+ Metall: A Case Study of Accelerating Graph Analytics Using Persistent Memory Allocator. https://github.com/Exa-Graph/miniVite/tree/metallds2





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

```
git clone https://github.com/ECP-ExaGraph/miniVite
```

Ghosh S, Halappanavar M, Tumeo A, Kalyanaraman A, Gebremedhin AH. miniVite: A graph analytics benchmarking tool for massively parallel systems. In 2018 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS) 2018 Nov 12 (pp. 51-56). IEEE.

Pacific Northwest
NATIONAL LABORATORY

sayan.ghosh@pnnl.gov

Thank you

```
Possible options (can be combined):

1. -f <bin-file>   : Specify input binary file after this argument.
2. -b              : Only valid for real-world inputs. Attempts to distribute approximately
                     equal number of edges among processes. Irregular number of vertices
                     owned by a particular process. Increases the distributed graph creation
                     time due to serial overheads, but may improve overall execution time.
3. -n <vertices>   : Only valid for synthetically generated inputs. Pass total number of
                     vertices of the generated graph.
4. -l              : Use distributed LCG for randomly choosing edges. If this option
                     is not used, we will use C++ random number generator (using
                     std::default_random_engine).
5. -p <percent>    : Only valid for synthetically generated inputs. Specify percent of overall
                     edges to be randomly generated between processes.
6. -t <threshold>  : Specify threshold quantity (default: 1.0E-06) used to determine the
                     exit criteria in an iteration of Louvain method.
7. -w              : Only valid for synthetically generated inputs. Use Euclidean distance as edge weight.
                     If this option is not used, edge weights are considered as 1.0. Generate
                     edge weight uniformly between (0,1) if Euclidean distance is not available.
8. -r <nranks>     : This is used to control the number of aggregators in MPI I/O and is
```

Ghosh S, Halappanavar M, Tumeo A, Kalyanaraman A, Lu H, Chavarria-Miranda D, Khan A, Gebremedhin A. Distributed louvain algorithm for graph community detection. In2018 IEEE international parallel and distributed processing symposium (IPDPS) 2018 May 21 (pp. 885-895). IEEE.

Ghosh S, Halappanavar M, Tumeo A, Kalyanaraman A, Gebremedhin AH. miniVite: A graph analytics benchmarking tool for massively parallel systems. In 2018 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS) 2018 Nov 12 (pp. 51-56). IEEE.

Ghosh S, Halappanavar M, Tumeo A, Kalyanaraman A, Gebremedhin AH. Scalable distributed memory community detection using vite. In2018 IEEE High Performance extreme Computing Conference (HPEC) 2018 Sep 25 (pp. 1-7). IEEE.

Ghosh S, Halappanavar M, Tumeo A, Kalyanarainan A. Scaling and quality of modularity optimization methods for graph clustering. In2019 IEEE High Performance Extreme Computing Conference (HPEC) 2019 Sep 24 (pp. 1-6). IEEE.

Gawande N, Ghosh S, Halappanavar M, Tumeo A, Kalyanaraman A. Towards scaling community detection on distributed-memory heterogeneous systems. Parallel Computing. 2022 Jul 1;111:102898.