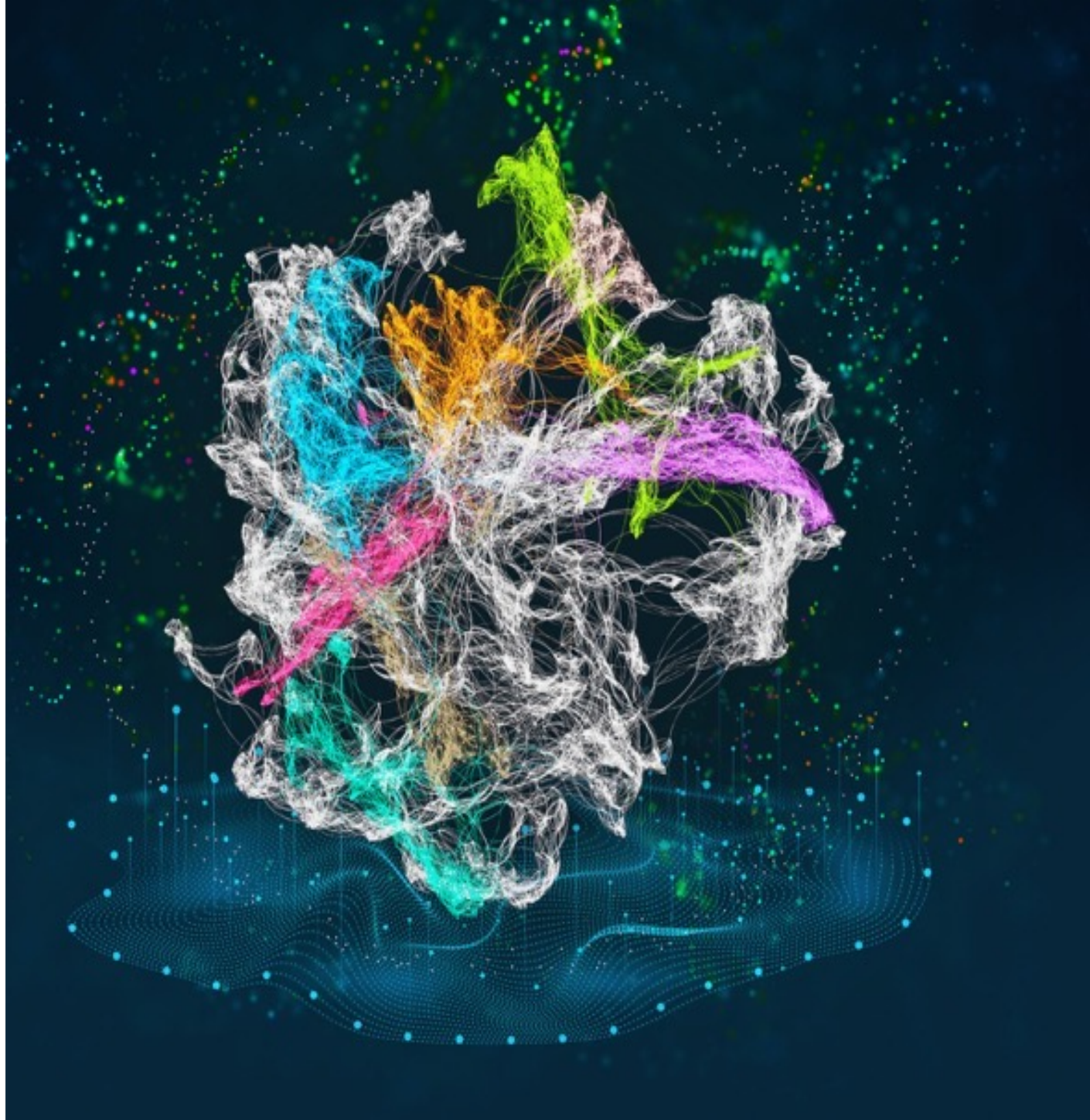
Pacific Northwest NATIONAL LABORATORY

EXASCALE COMPUTING PROJECT

Graph Analytics in the accelerator-enabled *Exascale* Era

Mahantesh Halappanavar on behalf of the ExaGraph Team

02 Nov 2022
SERDP All-hands



Acknowledgements

ExaGraph Core Team

- [Nitin Gawande]
- Sayan Ghosh
- Ananth Kalyanaraman
- Ariful Khan
- Marco Minutoli
- Kaisa Swirydowicz
- Nathan Tallent
- Antonino Tumeo

PNNL Partners

- Arun Sathanur
- Sam Chatterjee
- Jason McDermott
- Sam Silva
- Abhishek Somani
- Ajay Panyala
- S. Krishnamoorthy
- …

External Collaborators

- Alex Pothen (Exagraph)
- Aydin Buluc (Exagraph)
- Erik Boman (Exagraph)
- Prathyush Sambathuru
- Anil Vullikanti
- Sara Selitsky
- …

Research Interns

- Hao Lu
- Md Naim
- Tony Liu
- Ferdous SM
- Ahammed Ullah
- Sid Das
- Reet Barik
- Sriram S.
- Rounak M
- …

U.S. DEPARTMENT OF ENERGY

EXASCALE COMPUTING PROJECT

www.exagraph.org

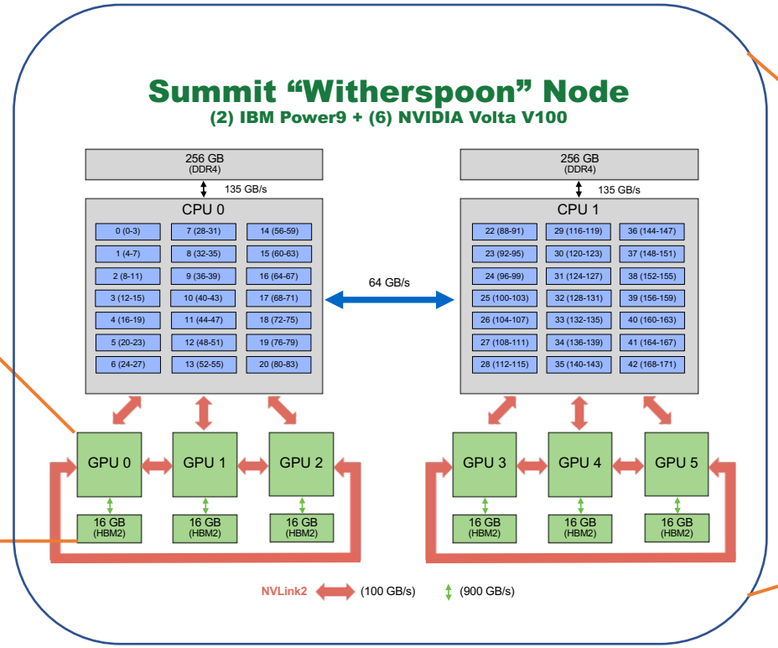
What is *exascale*?

10^{18}

Pre-exascale

Summit: Pre-exascale (IBM+Nvidia)

Single GPU
(2048 x 80) threads)

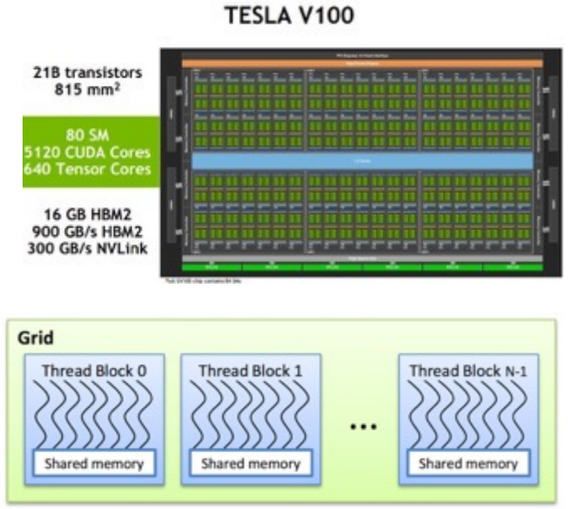


Single Node
(6 GPUs)

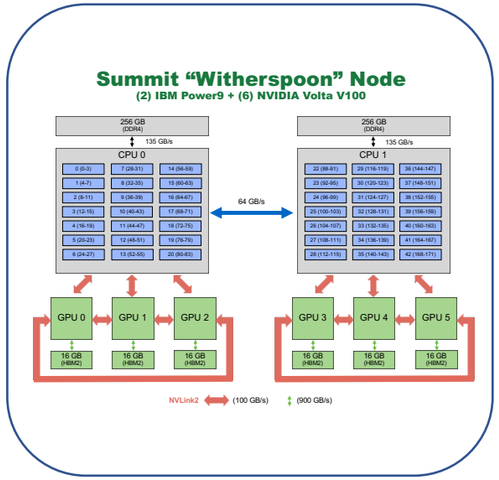
Distributed Multi-GPU Cluster
(4608 nodes)

$$\frac{\mathbf{2048}\ \text{Threads}}{\text{SM}} \times \frac{\mathbf{80}\ \text{SMs}}{\text{GPU}} \times \frac{\mathbf{6}\ \text{GPUs}}{\text{Node}} \times \mathbf{4608}\ \text{Nodes} = \mathbf{4.53\ Billion}\ \text{GPU Threads}$$

Significant Challenges



- Load balancing
- Sparse & irregular memory accesses
- Coalesced memory accesses
- SIMD & thread divergence



- Load balancing
- Work-division between host and GPU(s)
- Deep memory hierarchies (unified memory)
- Data movement

- Load balancing
- Communication and computation balance
- Complicated programming models (MPI+…+…)
- Data movement

Influence maximization

- Algorithms
- Applications
- Software

Image Credit: https://blog.edmentum.com/making-social-network-work-school

The influence maximization problem

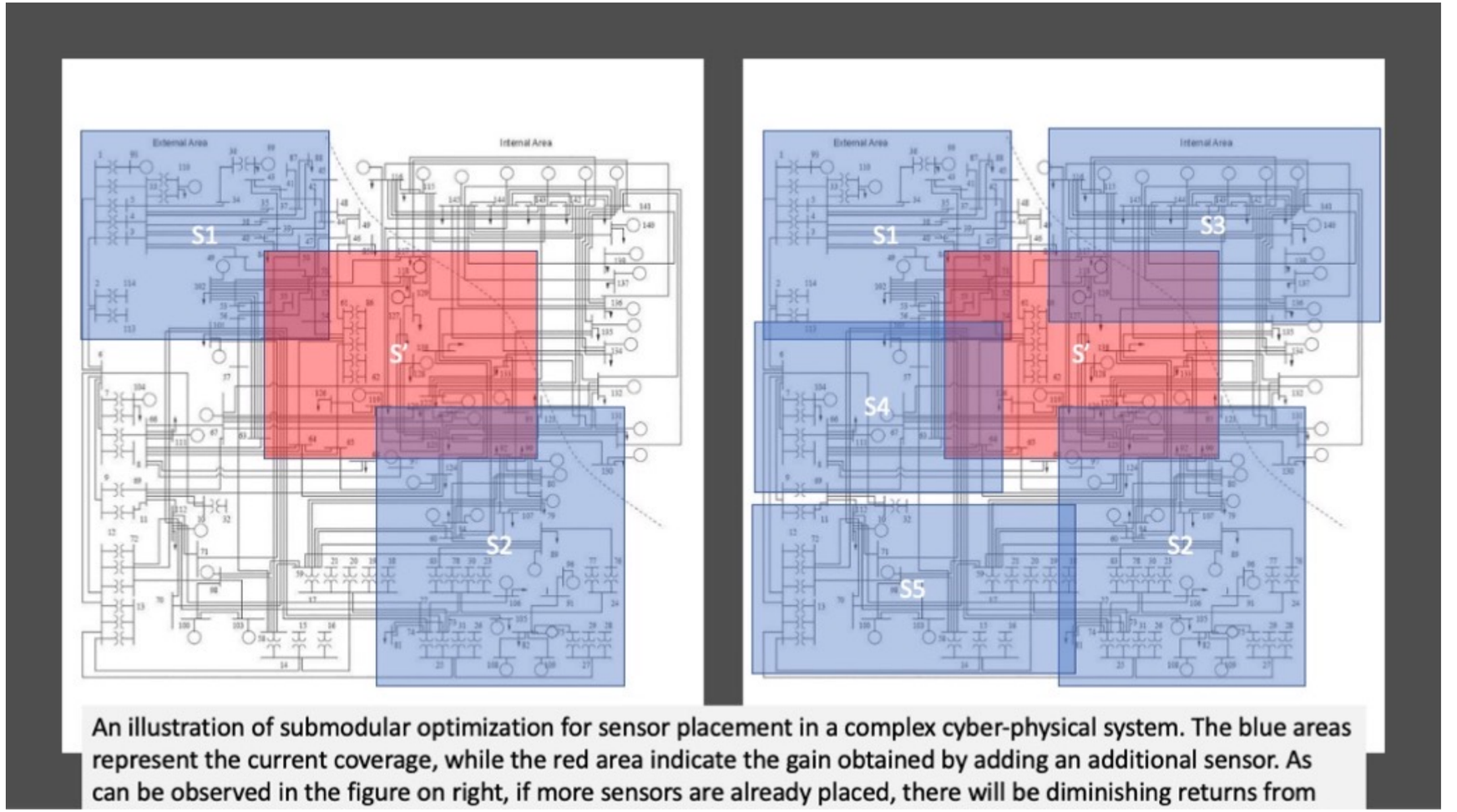
- Given: A graph $G=(V, E, \omega)$, a diffusion model (how a vertex gets activated based on the state of its neighbors), and a budget k, the influence maximization problem is stated as follows:

Find a set of k vertices called the seed set S, that when activated results in maximal activations in the network amongst all possible sets of k vertices

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

- Two diffusion models studied in our work:
  - Linear Threshold: A vertex can get activated if a fraction of neighboring vertices that are active is greater than a threshold Θ_v
  - Independent Cascade: One shot chance for an activated vertex to activate its neighbor

Submodularity: An illustrative example



An illustration of submodular optimization for sensor placement in a complex cyber-physical system. The blue areas represent the current coverage, while the red area indicate the gain obtained by adding an additional sensor. As can be observed in the figure on right, if more sensors are already placed, there will be diminishing returns from

Optimal:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Greedy Hill Climbing: Key steps

1. Generate a set of n random samples SG
   - Different instantiations of G are computed based on the edge probabilities

1

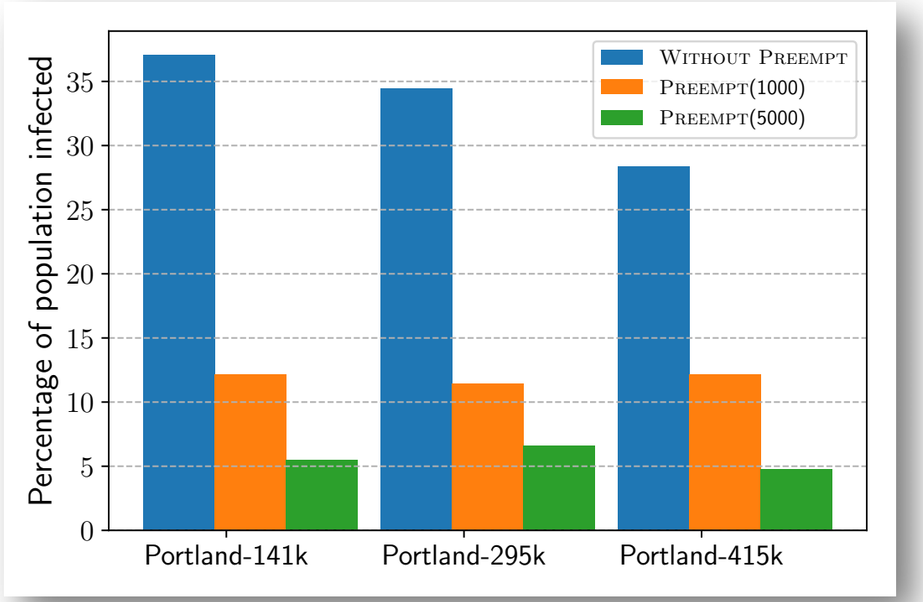
2. Repeat until k most influential nodes are chosen:
   1. Compute the influence of a chosen node across different samples w.r.t. the current seed set S
   2. Pick the best influential node, and add to S

2

- Key algorithmic difference between Linear Threshold and Independent Cascade algorithms arise in Step 1 (generation of random samples)
- Approximation Factor: (1 - 1/e) - **ε** (submodularity)

EpiControl: Controlling epidemic spread

- EpiControl: Given a graph G, a set of initially infected nodes B, and a budget k, find a set of nodes $S \subseteq V$ to vaccinate, such that $|S| \leq k$ and $\mathbb{E}[\lambda(S)]$ is maximized, where $\lambda(S)$ represents the number of lives saved
- Preempt: Given a graph G and budget k, find a set of nodes $S \subseteq V$ to vaccinate, such that $|S| \leq k$ and $\mathbb{E}[\sigma(S)]$ is maximized, treating S as the initial set of activated nodes, and where $\sigma(S)$ represents the reachability of S in G
- EpiControl on trees is submodular



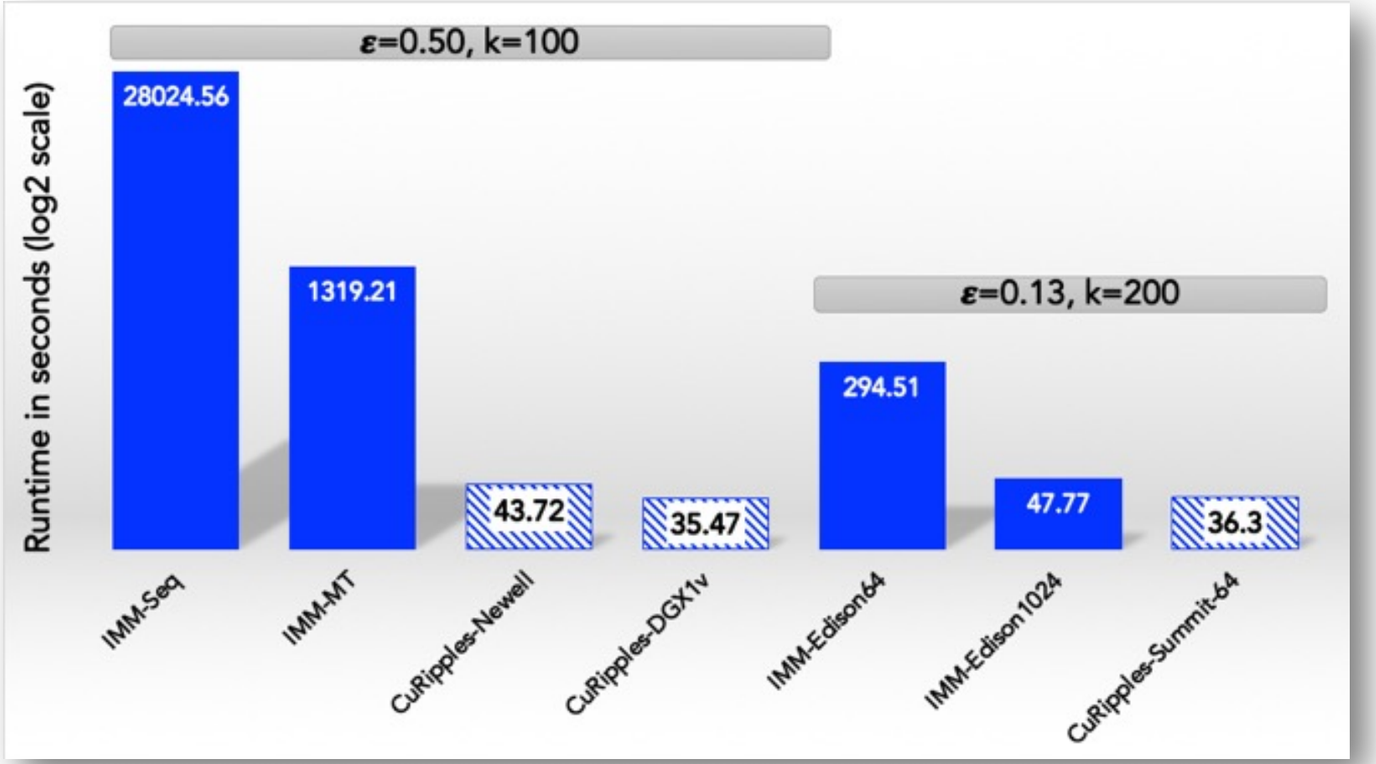
A comparison of the percentages of population infected with and without our proposed method Preempt, for three contact networks of Portland.

Even with relatively low budgets for vaccination (1000 and 5000 nodes), we obtain anywhere between 2.61x to 6.75x reduction in the percentages of reduction without Preempt.

Acknowledgements: Sambaturu, Vullikanti, et al.

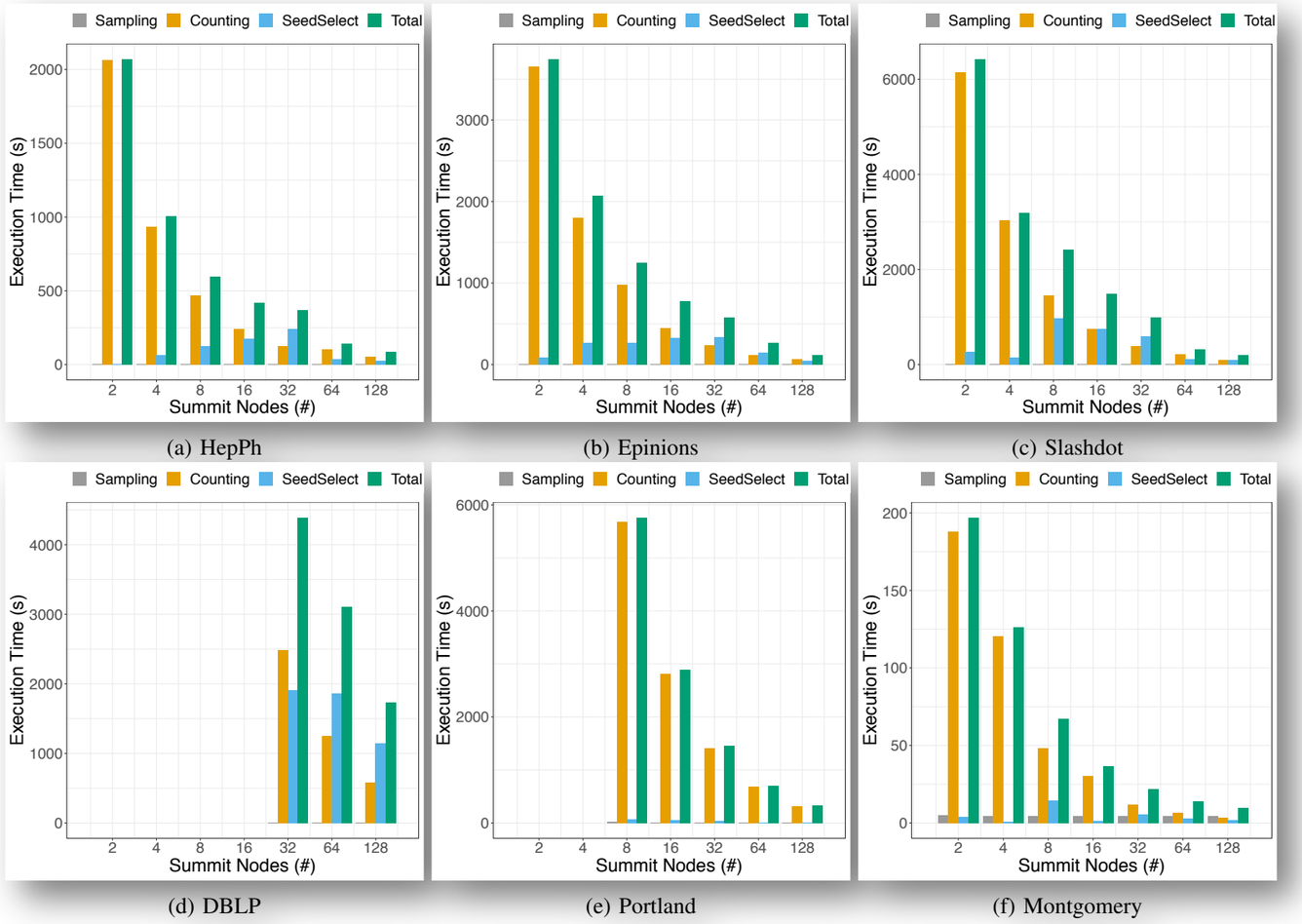
Ripples & cuRipples

Scalable implementations (shared and distributed memory systems)
https://github.com/pnnl/ripples



CuRipples achieves a speedup of 790x over a state-of-the-art serial implementation, while also significantly improving the quality.
The input network is com-Orkut.

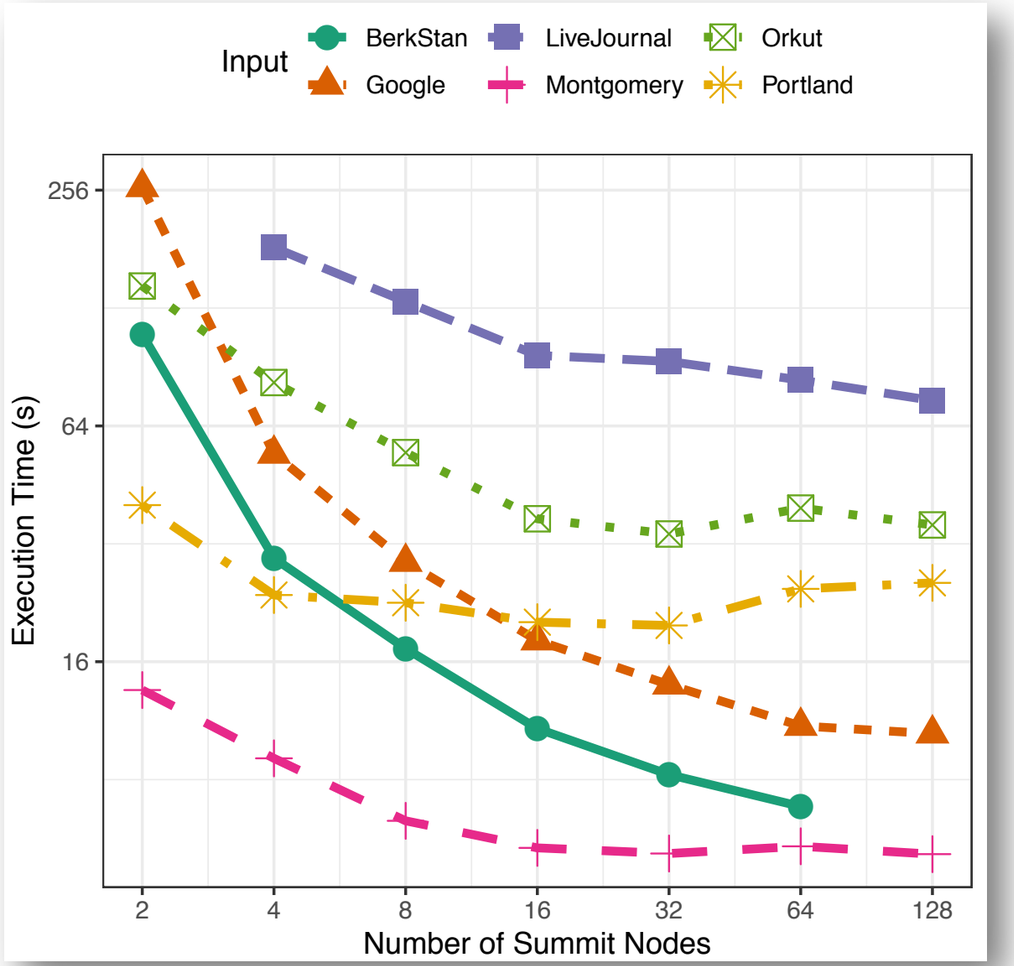
Strong scaling on Summit (Hill Climbing)



- Counting step dominates total time, and scales well -- large speedups (e.g., Slashdot: 63x, Montgomery: 155x)
- Strong scaling: Speedups for 128 nodes between 20x and 33x relative to two nodes of Summit
- Significantly reduces time to solution from hours to minutes (Slashdot: 1.8 hours on two Summit nodes vs. 3.2 minutes on 128 Summit nodes)

Strong Scaling using SNAP networks (a-d) and the Contact Networks (e-f). Missing data points are executions that did not complete under two hours.

Strong scaling on Summit (Rev Reachable)



Strong Scaling using SNAP networks

TABLE II: Comparative evaluation of `cuRipples` relative to previous implementations of IMM—both serial (IMM_{seq}) [2] and parallel ($IMM_{opt/mt/edison}$) [3]. Abbreviations used: No. Cores (C), GPUs (G), Nodes (N).

System	Time (s)	Speedup	Scale
com-Orkut (ϵ=0.5,k=100)			
IMM_{seq}	28024.56	1.00×	1C
IMM_{opt}	9027.50	3.10×	1C
IMM_{mt}	1319.21	21.24×	20C (1N)
$CuRipples_{dgx\text{-}1v}$	35.47	**790.09**×	80C+8G (1N)
$CuRipples_{newell}$	43.72	641.00×	128C+4G (1N)
com-Orkut (ϵ=0.13,k=200)			
IMM_{edison}	294.51	95.16×	3,072C (64N)
IMM_{edison}	47.77	586.61×	49,152C (1024N)
$CuRipples_{summit}$	36.30	**772.03**×	2,688C+384G (64N)
soc-LiveJournal1 (ϵ=0.5,k=100)			
IMM_{seq}	16434.81	1.00×	1C
IMM_{opt}	3954.59	4.16×	1C
IMM_{mt}	1026.21	16.02×	20C
$CuRipples_{dgx\text{-}1v}$	70.23	234.01×	80C+8G (1N)
$CuRipples_{newell}$	65.26	251.84×	128C+4G (1N)
soc-LiveJournal1 (ϵ=0.13,k=200)			
IMM_{edison}	190.94	86.07×	3,072C (64N)
IMM_{edison}	55.12	298.16×	49,152C (1024N)
$CuRipples_{summit}$	106.43	154.42×	2,688C+384G (64N)

References: Inf max

- [PREEMPT] M Minutoli, M Halappanavar, A Tumeo, A Kalyanaraman, A Vullikanti, and P Sambaturu. "PREEMPT: Scalable Epidemic Interventions Using Submodular Optimization on Multi-GPU Systems." In ACM/IEEE *International Conference for High Performance Computing*, Network, Storage and Analysis (SC'20). Nov 9--19, 2020. Held virtually.
- [Ripples] M Minutoli, M Halappanavar, A Kalyanaraman, A Sathanur, R Mcclure, J McDermott. "Fast and scalable implementations of influence maximization algorithms." In Proceedings of the *IEEE Cluster conference (*CLUSTER'19*)*, 12 pages, 2019.
- [cuRipples] M. Minutoli, M Drocco, M Halappanavar, A Tumeo, and A Kalyanaraman. "CuRipples: Influence Maximizationon Multi-GPU Systems," In proceedings of the *International Conference on Supercomputing* (ICS20), June 29 - July 2, 2020 in Spain.
- [Cascade] Halappanavar M, A Sathanur, and A Nandi. "Accelerating the Mining of Influential Nodes in Complex Networks through Community Detection." In proceedings of the ACM *International Conference on Computing Frontiers*. May 16 - 18, 2016. Italy.
- A Sathanur, M Halappanavar, Y Shi, and Y Sagduyu. "Exploring the Role of Intrinsic Nodal Activation on the Spread of Influence in Complex Networks." in *Lecture Notes in Social Networks* (LNSN). 2018.
- U Bhatia, S Chatterjee, A Ganguly, M Halappanavar, R Tipireddy, and R Brigantic. "Aviation Transportation, Cyber Threats, and Network-of-Networks: Conceptual Framing and Modeling Perspectives for Translating Theory to Practice." Accepted for publication in proceedings of the *IEEE International Symposium on Technologies for Homeland Security* (HST). October 23 - 24, 2018 Woburn, MA.
- A Sathanur, M Halappanavar, S Chaterjee, A Ganguly, and K Clark, "Identification of Critical Airports from the Perspective of Delay and Disruption Propagation in Air Travel Networks", *IEEE Symposium on Technologies for Homeland Security*, Boston, Nov 2019
- A Sathanur, and M Halappanavar. "Influence Maximization on Complex Networks with Intrinsic Nodal Activation." Accepted for publication in proceedings of the *8th International Conference on Social Informatics* (SocInfo). Bellevue, WA, USA. November 2016.

Community detection

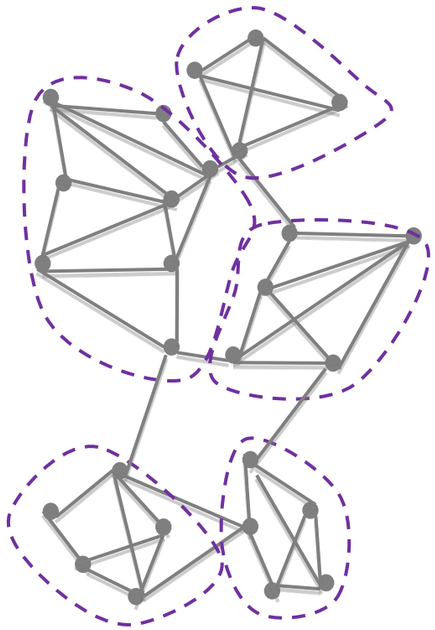
- Algorithms
- Applications
- Software

Graph Clustering

- Problem: Given $G=(V, E, \omega)$, identify tightly knit groups (clusters) 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

$P = \{C_1, C_2, \ldots C_k\}$

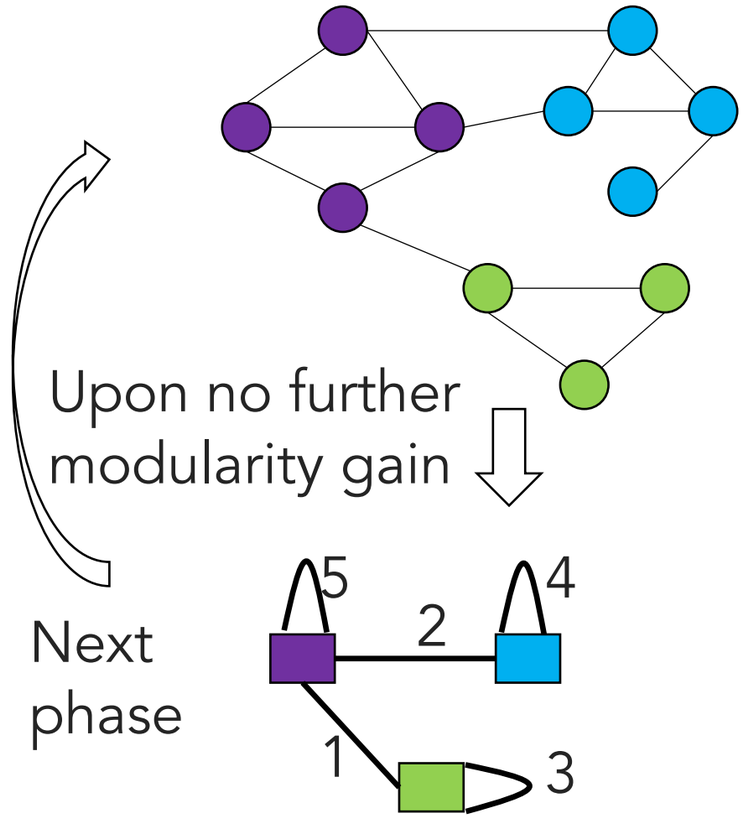
such that: …

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}*)

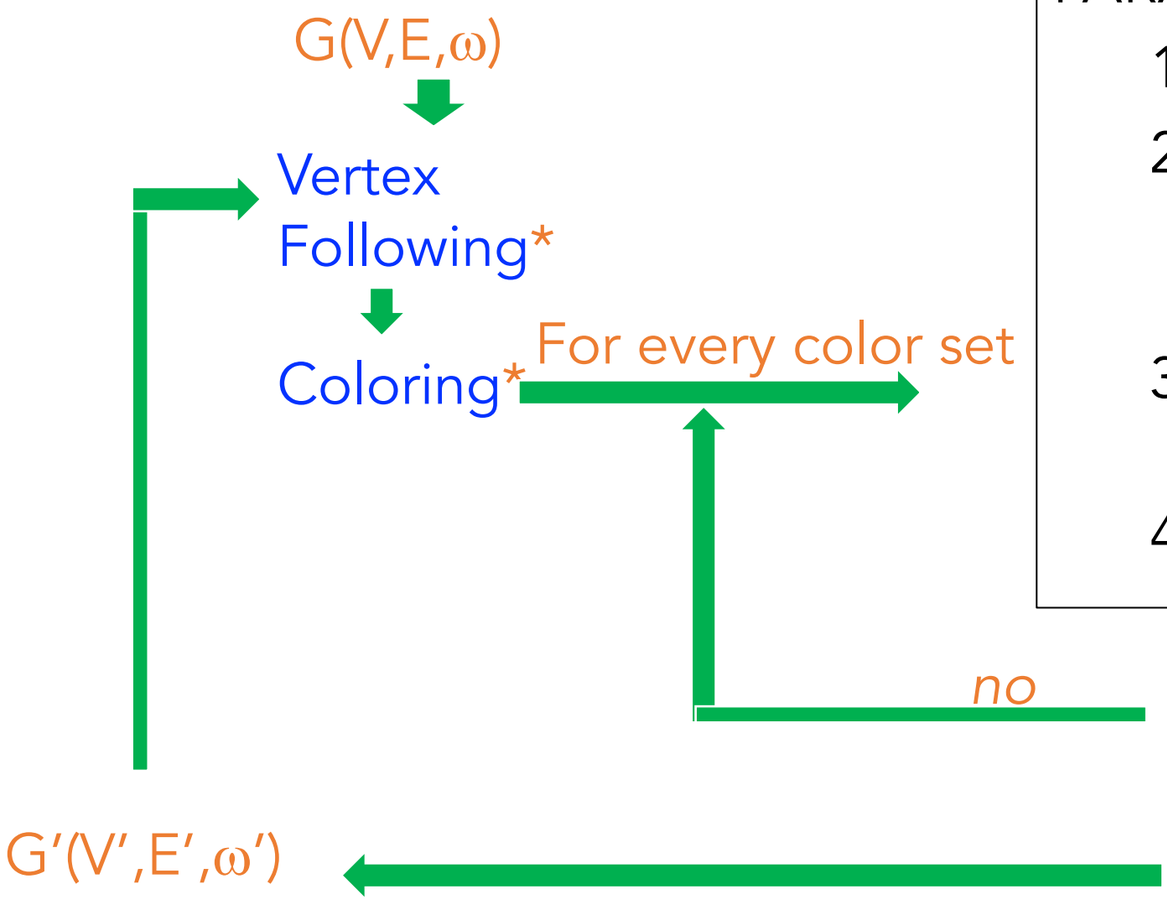


Multi-phase multi-iterative heuristic

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

Check for convergence

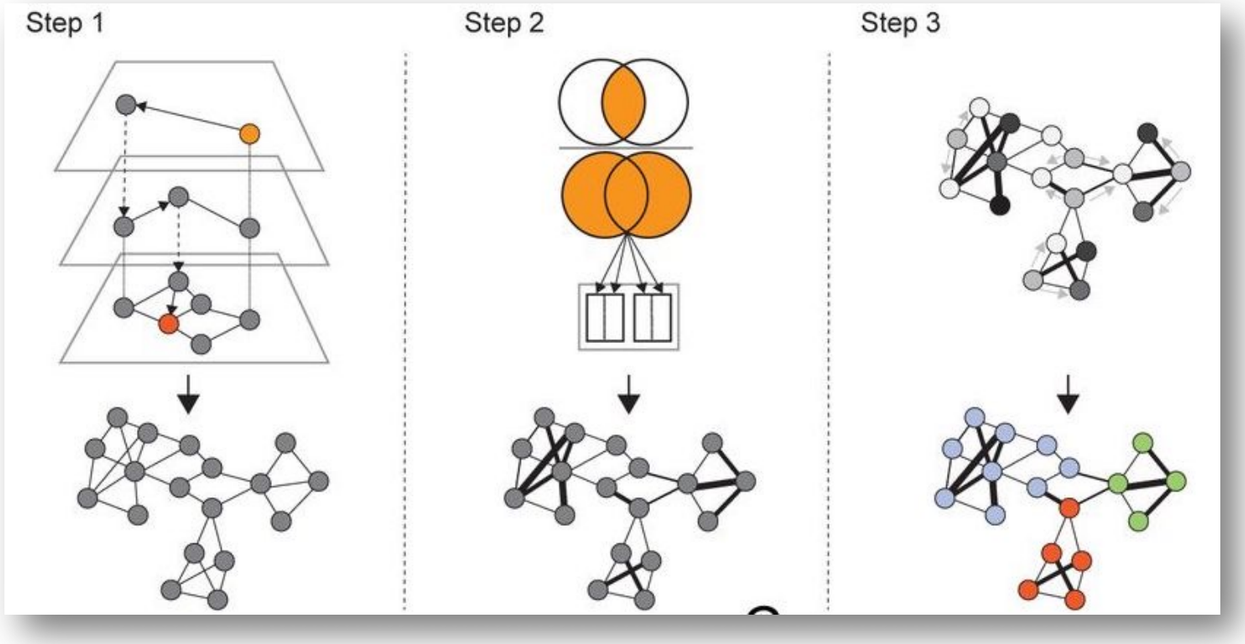
yes

Transform graph by collapsing communities into nodes

* Steps are optional

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

FastPG: Fast clustering of millions of single cells



Step 1. kNN-approximation HNSW, which has logarithmic scaling due to the hierarchical structure of the search space (depicted). The output of this step is a network of cells, where each node is a cell and neighbor are connected by an edge.

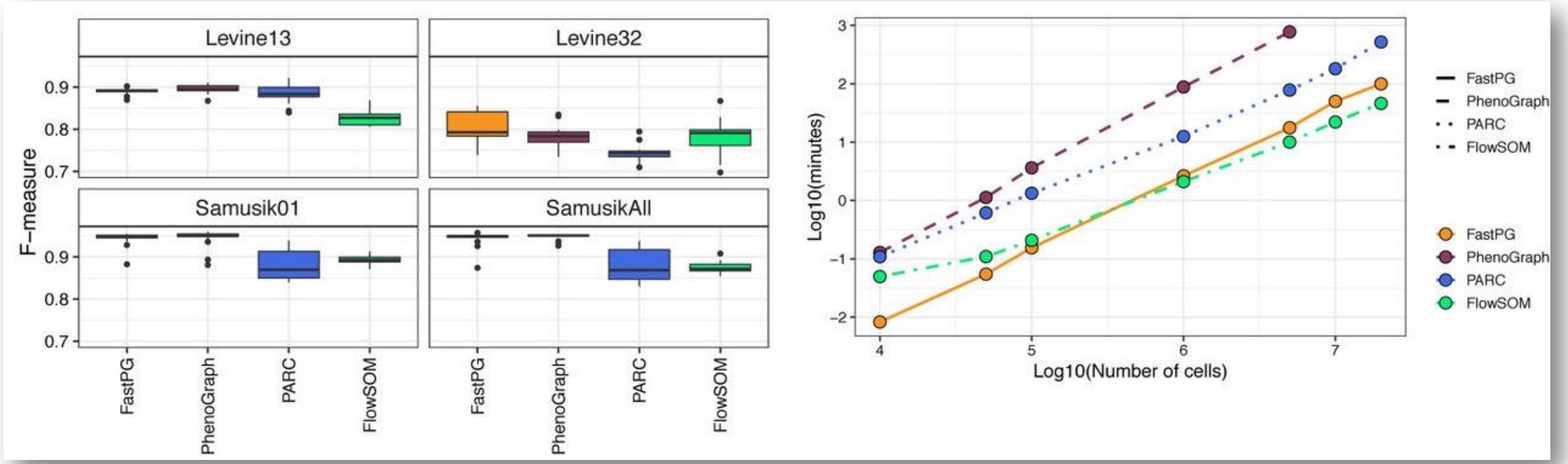
Step 2. Modification of the Jaccard index step to run in parallel, depicted as being distributed to each thread of the CPU. This step adds weights to the network, which are represented as different edge thicknesses.

Step 3. Grappolo: The output of this step is the assignment of cells to communities, which was depicted with different colored nodes.

Lead: Sara Selitsky

https://www.biorxiv.org/content/10.1101/2020.06.19.159749v2

FastPG: “Gold standard” datasets



(Left) Boxplot displaying the F-measure for four mass cytometry “gold standard” datasets.

(Right) Runtime comparisons between PhenoGraph, FastPG, PARC, and FlowSOM.

https://www.biorxiv.org/content/10.1101/2020.06.19.159749v2

Tools

- Grappolo: Scalable multi-threaded implementation using OpenMP
- Rundemanen: Scalable single-GPU implementation using CUDA
- Vite: Scalable distributed-memory implementation using MPI+OpenMP
- cuVite: Scalable distributed-memory implementation MPI+OpenMP+CUDA
- miniVite: Simplified variant of Vite for benchmarking (ECP Proxy App)

Photograph graciously provided by Jagan Bontha.

Grappolo

A **multithreaded** C++ and OpenMP library for graph clustering (community detection) based on the Louvain method. Several heuristics including distance-1 coloring and vertex following are used to parallelize the Louvain method efficiently.

Click this link to download the source code (08-2015)

Email Mahantesh for the latest version (will be made available soon)

Developers: Mahantesh Halappanavar and Howard (Hao) Lu with contributions from Sayan Ghosh, Ananth Kalyanaraman and Daniel Chavarria

License: BSD 3-Clause license (**Open Source Initiative**).

Vite

A **distributed-memory** implementation of Grappolo in C++ using MPI and OpenMP.

Click this link to download the source code (03-2018)

Developers: Sayan Ghosh, Daniel Chavarria and Antonino Tumeo with contributions from Hao Lu, Mahantesh Halappanavar, Ananth Kalyanaraman and Assefaw Gebremedhin
License: BSD 3-Clause license (**Open Source Initiative**).

miniVite

A simplified version of Vite for benchmarking purposes, targeting **distributed-memory** platforms using MPI and OpenMP. Part of **ECP Proxy Applications Suite.**

Click this link to download the source code (09-2018)

Alternate link for download (Github)

Developers: Sayan Ghosh, Daniel Chavarria, Antonino Tumeo with contributions from Hao Lu, Mahantesh Halappanavar, Ananth Kalyanaraman and Assefaw Gebremedhin
License: BSD 3-Clause license (**Open Source Initiative**).

Rundemanen

CUDA C++ parallel program for community detection.

Developers: Md Naim (naim.md@gmail.com) and Fredrik Manne (Fredrik.Manne@uib.no)
University of Bergen

License: BSD 3-Clause license (**Open Source Initiative**)

http://hpc.pnl.gov/people/hala/grappolo.html

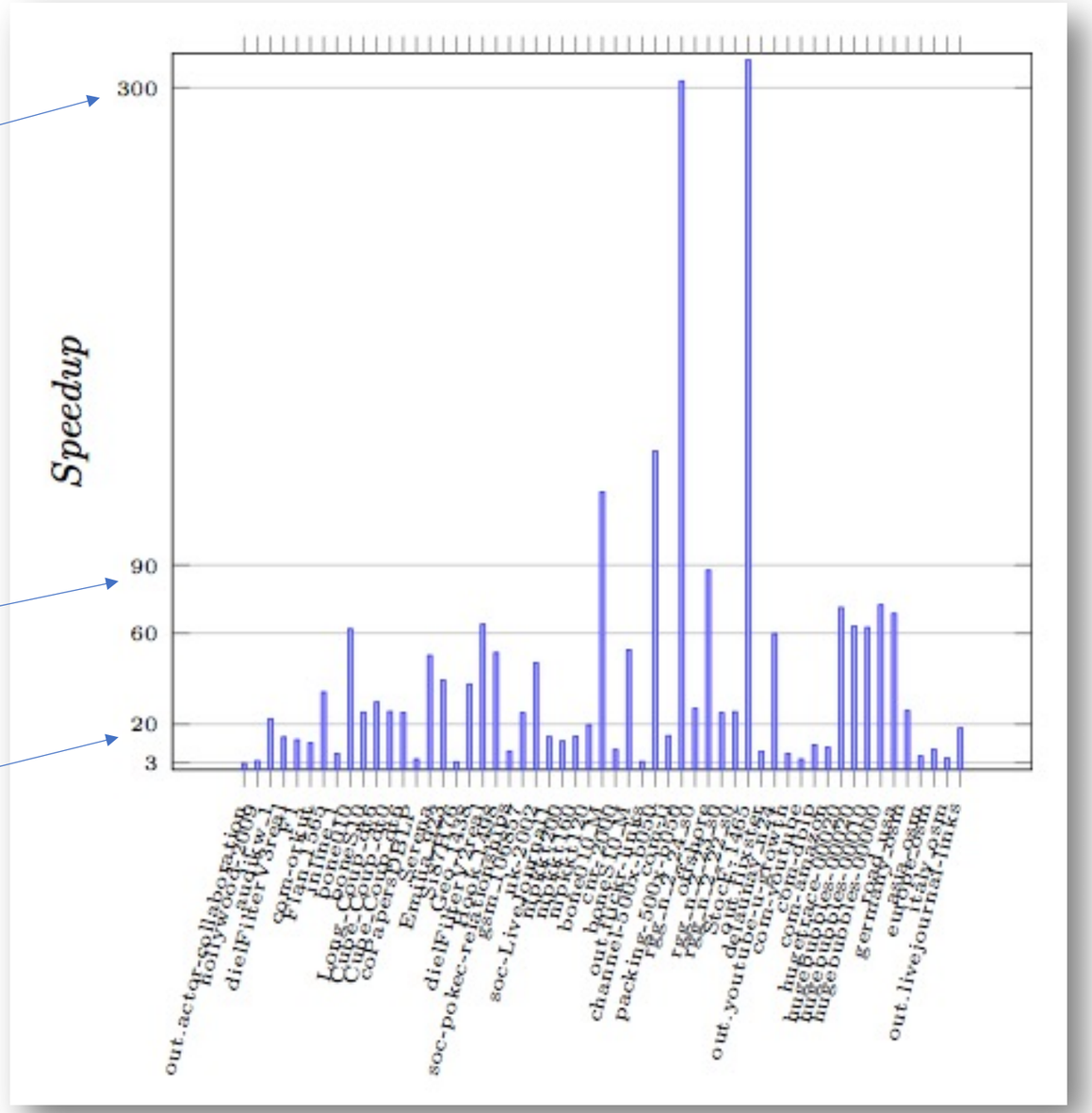
Single-GPU

Algorithmic innovation:

- Edge-centric parallelism implemented with GPU threads
- Load-balancing by bucketing of vertices that have nearly identical degree

Limitations:

- Maximum problem size limited by GPU memory
- Optimizations applicable for single GPU



Speedup w.r.t. sequential (Blondel et al.)

Naim, Manne, Halappanavar, and Tumeo. **Community Detection on the GPU**. In 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS) (pp. 625-634).

Distributed Grappolo: Vite

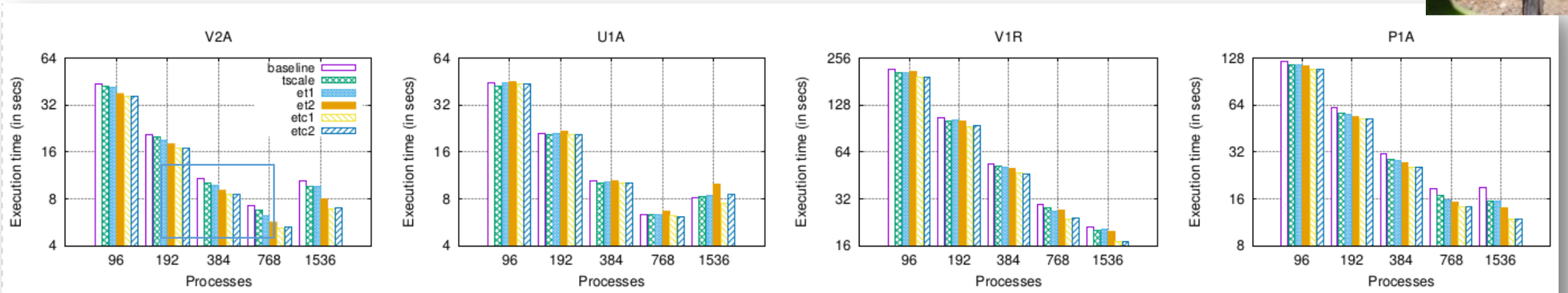


Fig. 7. Scalability of protein k-mer graphs.

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%

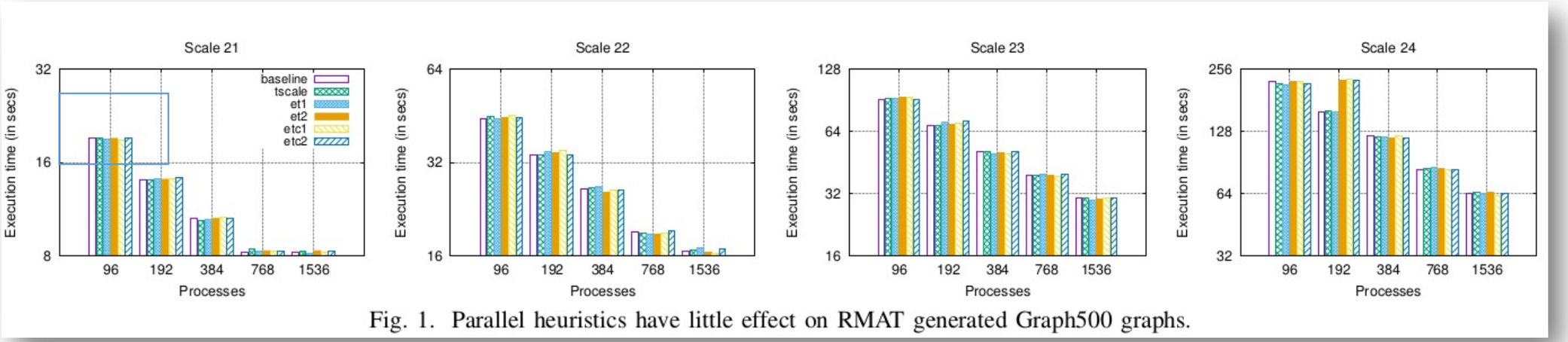
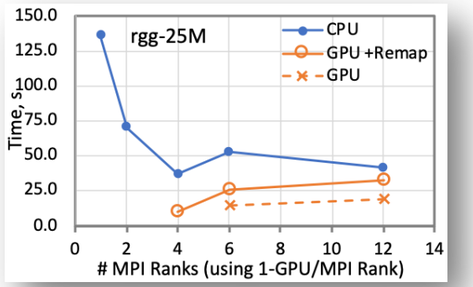


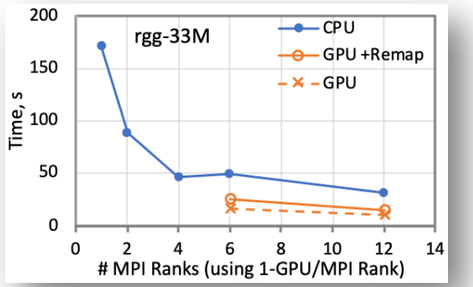
Fig. 1. Parallel heuristics have little effect on RMAT generated Graph500 graphs.

However, heuristics have little impact for some inputs!

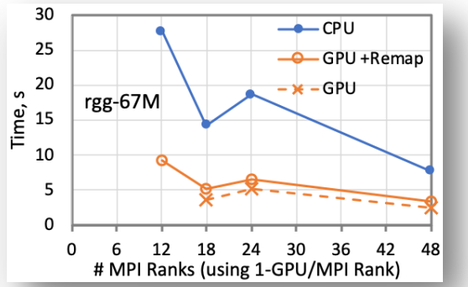
cuVite: Experiments on Summit



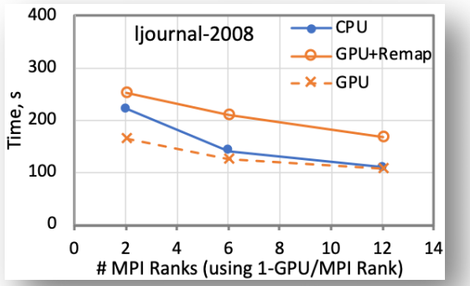
(a) 25 million vertices.



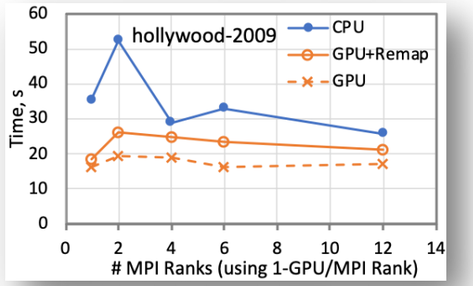
(b) 33 million vertices.



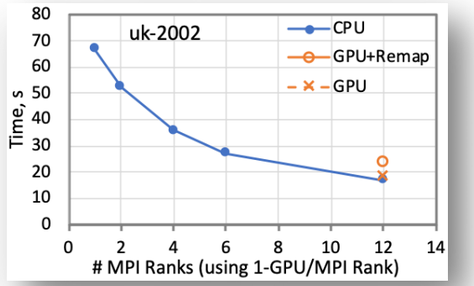
(c) 67 million vertices.



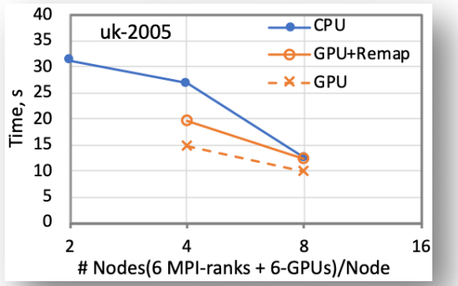
(a) ljournal dataset.



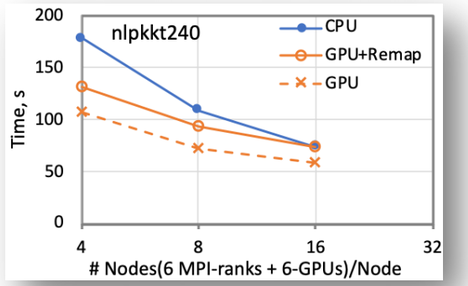
(b) Hollywood-2009 dataset.



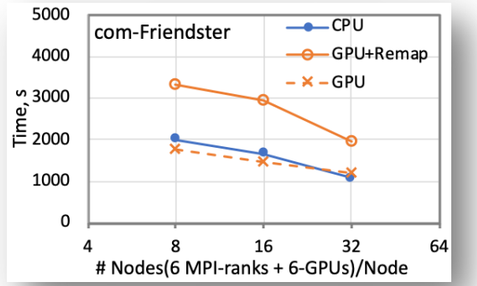
(c) UK-2002 dataset.



(a) uk-2005 dataset.



(b) nlpkkt-240 dataset.



(c) com-Friendster dataset.

- Preliminary results for strong scaling on up to 64 nodes on Summit
- Significant overhead due to data structure limitations
- Hybrid CPU-GPU code is harder to optimize due to load imbalances
- Several optimizations are planned for implementation

References: Community detection

- [Grappolo] H Lu, M Halappanavar, and A Kalyanaraman. "Parallel Heuristics for Scalable Community Detection." In *Elsevier Journal of Parallel Computing: Systems and Applications* (ParCo). Volume 47, August 2015, Pages 19--37. DOI: doi:10.1016/j.parco.2015.03.003. (Open access article).
- [Vite] S Ghosh, M Halappanavar, A Tumeo, A Kalyanaraman, H Lu, A Khan, and A Gebremedhin. "Distributed Louvain Algorithm for Graph Community Detection." In proceedings of the 32nd *IEEE International Parallel & Distributed Processing Symposium*. May 21 – May 25, 2018. Vancouver, Canada.
- [miniVite] S Ghosh, A Kalyanaraman, A Gebremedhin, M Halappanavar, A Tumeo. "miniVite: A Graph Analytics Benchmarking Tool for Massively Parallel Systems." *2018 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High-Performance Computer Systems* (PMBS). Dallas, TX, USA.
- [Rundemanen] M. Naim, F. Manne, M. Halappanavar and A. Tumeo, "Community Detection on the GPU," 2017 *IEEE International Parallel and Distributed Processing Symposium* (IPDPS), Orlando, FL, 2017, pp. 625-634.
- [Book] A Kalyanaraman, M Halappanavar, D Chavarría-Miranda, H Lu, K Duraisamy and P Pratim Pande. "Fast Uncovering of Graph Communities on a Chip: Toward Scalable Community Detection on Multicore and Manycore Platforms", *Foundations and Trends® in Electronic Design Automation*: Vol. 10: No. 3, pp 145-247. http://dx.doi.org/10.1561/1000000044
- S. Ghosh, M. Halappanavar, A. Tumeo, A. Kalyanaraman. "Scaling and quality of modularity optimization methods for graph clustering." In Proceedings of the *IEEE High Performance Extreme Computing* (HPEC'19), 6 pages, 2019. [**Amazon Graph Challenge Innovation Award**]
- D Chavarría-Miranda, A Panyala, M Halappanavar, J Manzano, and A Tumeo. "Optimizing Irregular Applications for Energy and Performance on the Tilera Many-core Architecture." In proceedings of the *ACM International Conference on Computing Frontiers*, Ischia, Italy. May 18-21, 2015.
- S. Ghosh, M. Halappanavar, A. Tumeo, A. Kalyanaraman and A. Gebremedhin, "Scalable Community Detection Using Vite." In proceedings of the 2018 *IEEE High Performance Extreme Computing Conference* (HPEC), Waltham, MA, 2018. [**Amazon Graph Challenge Student Innovation Award**]
- M. Halappanavar, H. Lu, A. Kalyanaraman and A. Tumeo, "Scalable static and dynamic community detection using Grappolo." In proceedings of the 2017 *IEEE High Performance Extreme Computing Conference* (HPEC), Waltham, MA, 2017, pp. 1-6. [**DARPA/Amazon Graph Challenge Champions**]

Other graph algorithms

- Matching or linear assignment problem
  - B-matching, submodular matching, covering, streaming, etc.
- Graph coloring
  - Distance-1 coloring, balanced coloring, partial distance-2 coloring, etc.
- Network alignment
  - Framework for using heuristics, subgraph isomorphism
- PageRank centrality computations
  - Approximate computing for scalability, Laplacian solver
- Chordal subgraph extraction

Pacific Northwest
NATIONAL LABORATORY

Thank you.

EXASCALE COMPUTING PROJECT

www.exagraph.org