Distributed-memory Graph Algorithms: Case studies with *Community Detection* and *Weighted Matching*

Sayan Ghosh*, Mahantesh Halappanavar+, Ananth Kalyanaraman*, Assefaw Gebremedhin*, Antonino Tumeo+

*Washington State University, Pullman, WA +Pacific Northwest National Laboratory, Richland, WA

About

- Distributed-memory graph applications exhibit irregular communication patterns, challenging to parallelize
- We study distributed-memory implementations of Community Detection (using *Louvain method*) and Maximum Weight Matching (half approximate method)
  - Partition a graph into *clusters* (or *communities*) such that each cluster consists of vertices that are densely connected within the cluster and sparsely connected to the rest of the graph
  - A *matching* in a graph is a subset of edges such that no two "matched" edges are incident on the same vertex

Louvain method for graph clustering

- Goodness of partitioning measured using a global metric called *modularity* (Q), that depends on sum of intra and inter community edge weights
- In 2008, Blondel, et al. introduced a multi-phase, iterative heuristic for modularity optimization, called the *Louvain method*

Initially each vertex assigned to a separate community

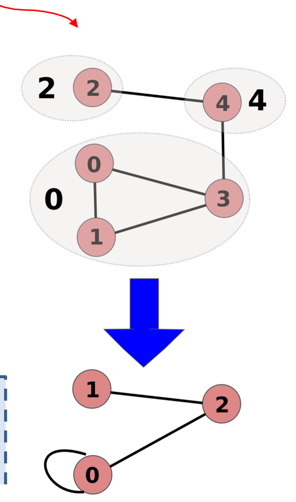
Within each iteration
- ΔQ when a vertex migrates
- Move vertex from current community to one that yields max ΔQ

At the end of a phase, the graph is rebuilt

Input: Graph $G = (V, E)$, threshold τ, Initial community assignment, C_{init}

$Q_{prev} \leftarrow -\infty$
$C_{prev} \leftarrow$ Initialize each vertex in its own community
while true **do**
  for all $v \in V$ **do**
    $N(v) \leftarrow$ neighboring communities of v
    $targetComm \leftarrow \arg\max_{t \in N_v} \Delta Q(v \text{ moving to } t)$
    if $\Delta Q > 0$ **then**
      Move v to $targetComm$ and update C_{curr}
  $Q_{curr} \leftarrow ComputeModularity(V, E, C_{curr})$
  if $Q_{curr} - Q_{prev} \leq \tau$ **then**
    break
  else
    $Q_{prev} \leftarrow Q_{curr}$

Phase continues until ΔQ between successive iterations is below a threshold



Maximum weight matching

- In the first phase, the initial set of locally dominant edges are identified and added to matching set M
- Next phase is iterative, for each vertex in M, its unmatched neighboring vertices are matched

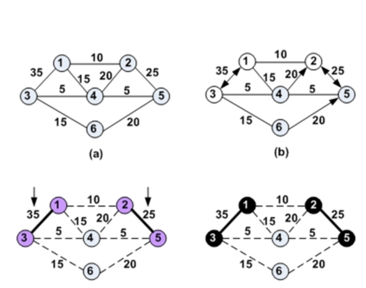
Input: Graph $G = (V, E, \omega)$.
Output: M set of matched vertices.

$mate_v \leftarrow \emptyset \quad \forall v \in V$
$M \leftarrow \emptyset$
for $v \in V$ **do**
  $u \leftarrow mate_v \leftarrow \arg\max_{u \in N'_v} \omega_{u,v}$
  if $mate_u = v$ **then**
    $M \leftarrow M \cup \{u, v\}$
while true **do**
  $v \leftarrow$ *some vertex from* M
  for $x \in N'_v$ *where* $mate_x = v$ *and* $x \notin M$ **do**
    $y \leftarrow mate_x \leftarrow \arg\max_{y \in N'_x}$
    if $mate_y = x$ **then**
      $M \leftarrow M \cup \{x, y\}$
  if *processed all vertices in* M
    break

N'$_v$ represents unmatched vertices in v's neighborhood

Vertex with the heaviest unmatched edge incident on v is referred as v's *mate*

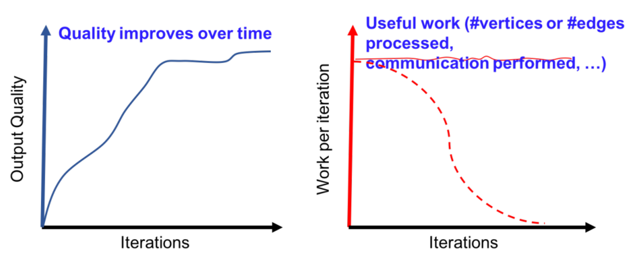
mate of a vertex can change as it may try to match with multiple vertices in its neighborhood



Heuristics for Community Detection

Objective: To devise heuristics that improve execution time performance and/or quality.

Early Termination

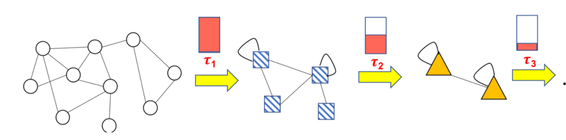


$$P_{v,k} = \begin{cases} P_{v,k-1} * (1-\alpha), & \text{if } C_{v,k-1} = C_{v,k-2} \\ 1, & \text{otherwise} \end{cases}$$

Penalize a vertex in every iteration if it stays in the same community, eventually it becomes immobile if the cumulative penalty fall below a cutoff (another variant requires global communication).

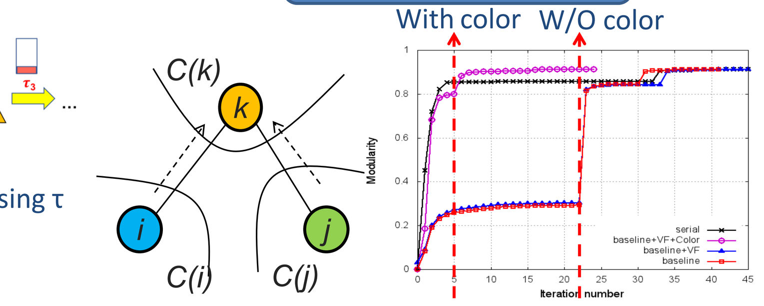
When α is close to 1, this scheme is more aggressive in terminating vertices, whereas close to 0 is the baseline case.

Threshold Cycling



Initially, when the graph is large, increasing τ leads to quicker exit per phase.

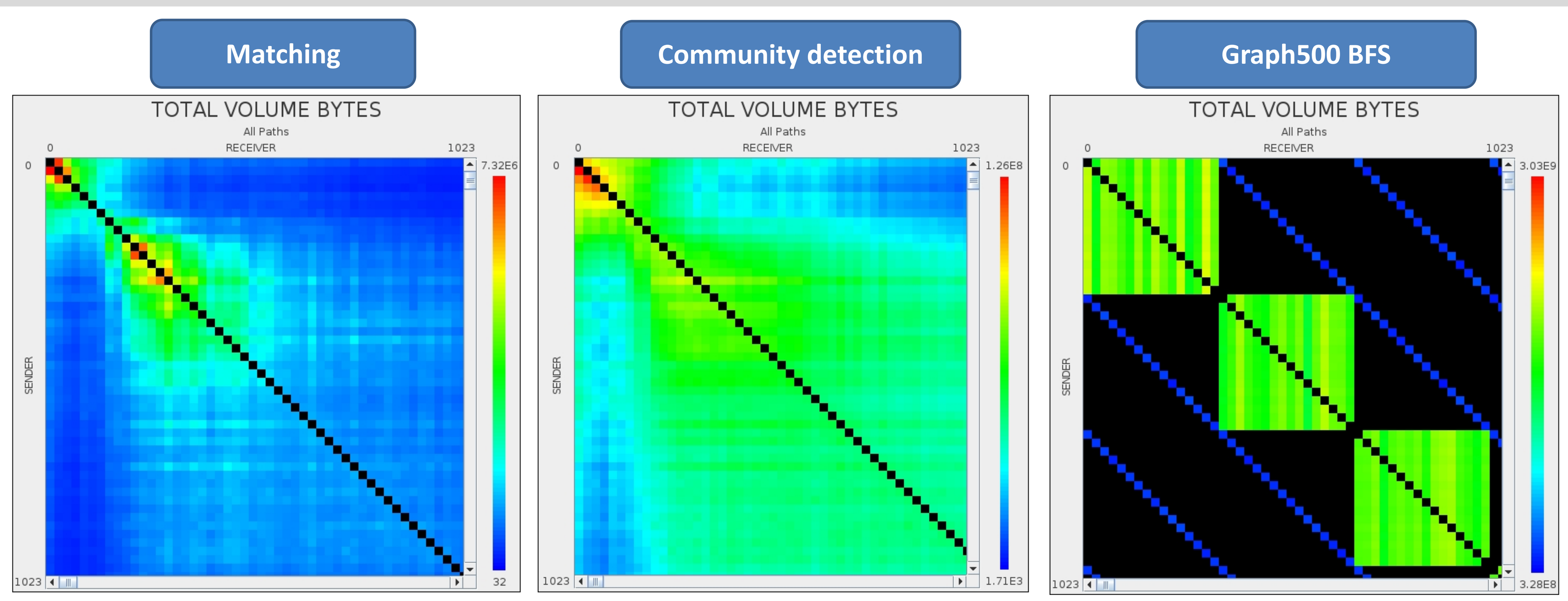
Incomplete coloring



Without coloring, in parallel a negative gain scenario is possible (since processes work with outdated information). We color a fraction of vertices with preselected number of color classes using the Jones-Plassmann algorithm.

Coloring is expensive in distributed memory as entire Louvain iteration needs to be invoked per color class, increasing the communication calls.

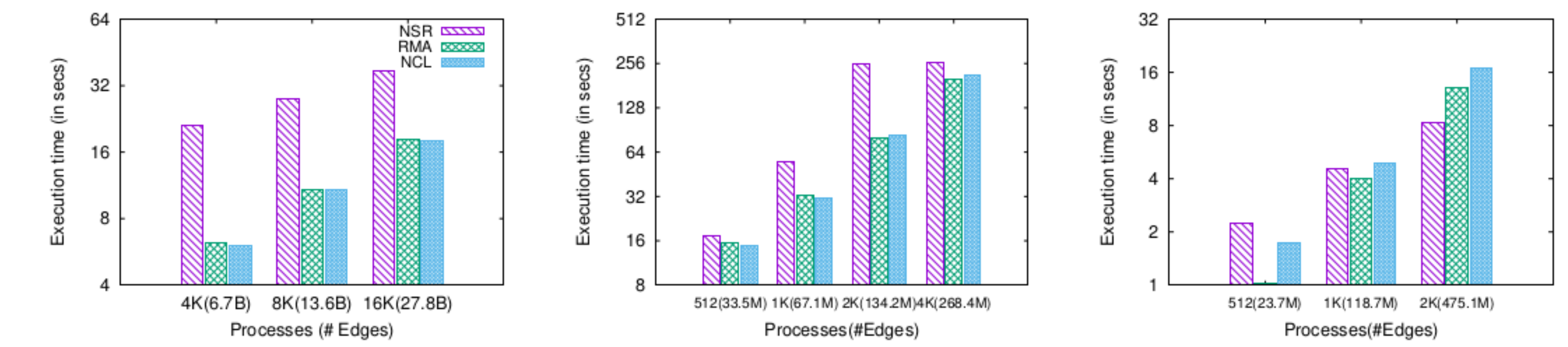
Communication characteristics on NERSC Cori (1K processes)



Friendster (1.8B edges)   R-MAT graph (2.1B edges)

Performance: Half approximate matching

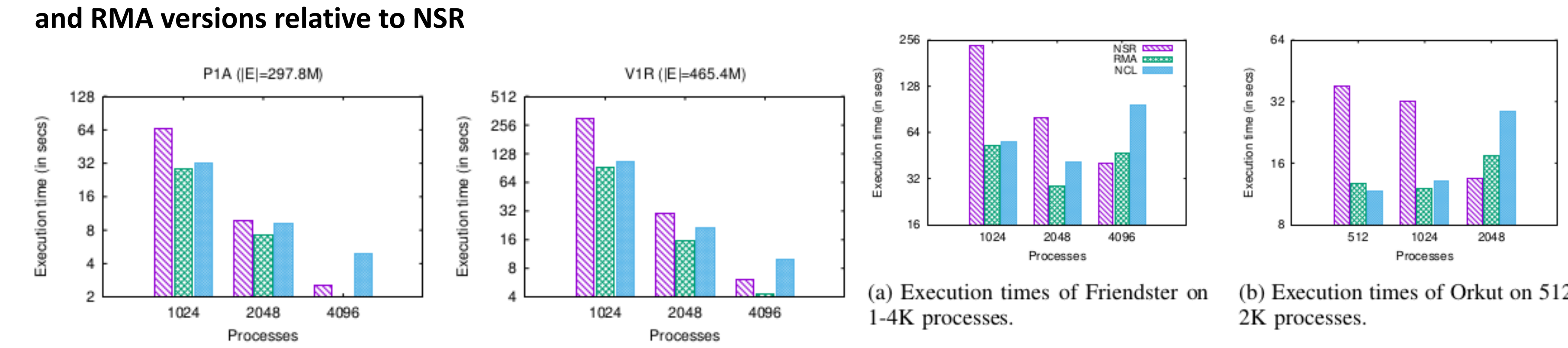
Objective: Implemented half-approx matching using MPI Send-Recv (NSR), Neighborhood collectives (NCL) and RMA.



(a) Random geometric graphs on 4-16K processes. (b) Graph500 R-MAT graphs on 512-4K processes. (c) Stochastic block-partitioned graph on 512-2K processes.

Observed 2-3.5x speedup on 4-16K processes for both NCL and RMA versions relative to NSR

NCL/RMA is not efficient for this input



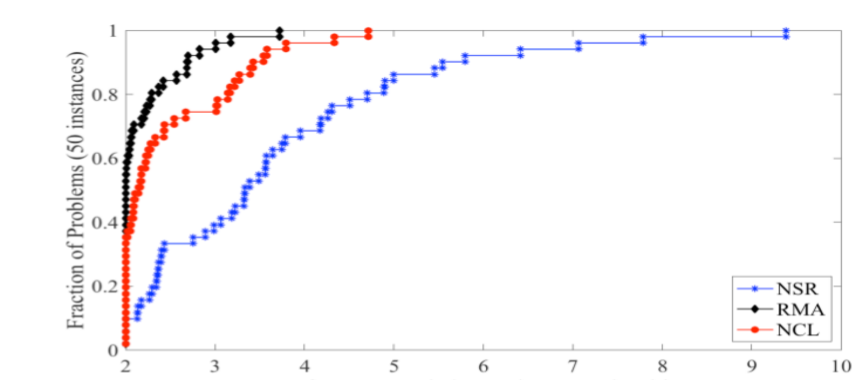
(a) Execution times of Friendster on 1-4K processes. (b) Execution times of Orkut on 512-2K processes.

RMA performs at least 25 -35% better than NSR and NCL

Large neighborhood results in poor performance

Graph category	Identifier	Best speedup	Version
Random geometric graphs (RGG)	d=8.56E-05	3.5×	NCL
	d=6.12E-05	2.56×	NCL
	d=4.37E-05	2×	NCL
Graph500 R-MAT	Scale 21	2.32×	NCL
	Scale 22	3×	RMA
	Scale 23	3.17×	RMA
	Scale 24	2×	NCL
Protein K-mer	V2a	1.4×	RMA
	U1a	2.2×	RMA
	P1a	2.32×	RMA
	V1r	3.3×	RMA
DNA	Cage15	**6**×	NCL
CFD	HV15R	4×	NCL
Social network	Orkut	3.26×	NCL
	Friendster	**4.45**×	RMA

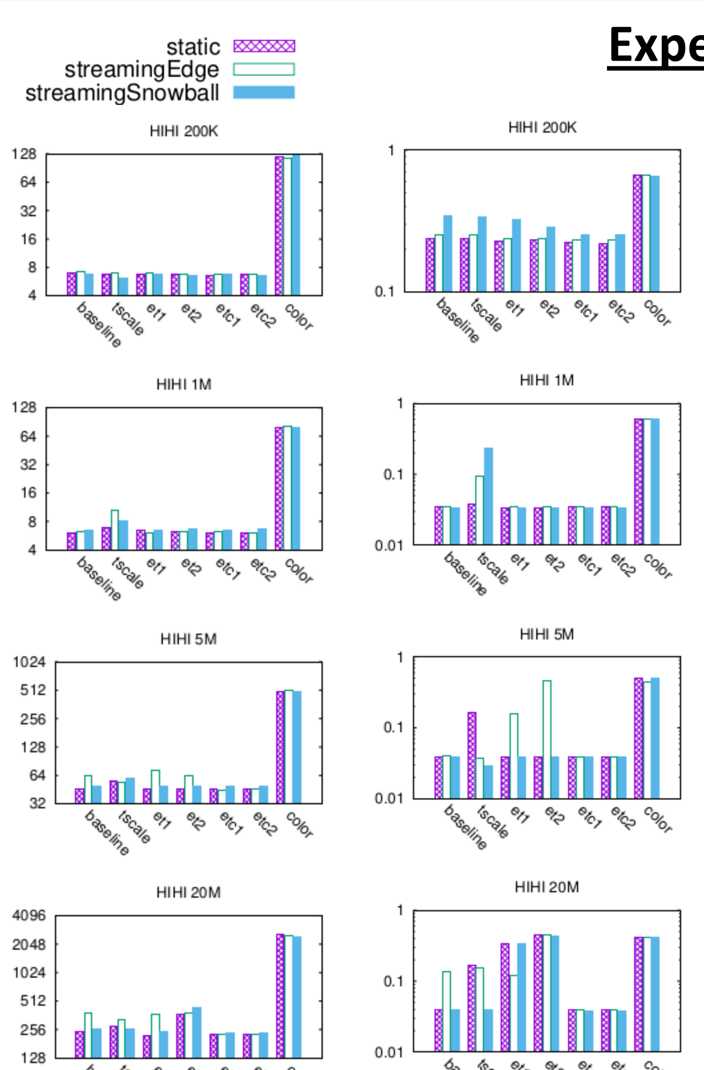
Versions yielding the best performance over the Send-Recv baseline version (run on 512-16K processes) for input graphs.



The relative performance profiles for NSR, RMA and NCL using a subset of inputs used in the experiments. The X-axis represents the factor by which a given scheme fares relative to the best performing scheme for that particular input. The Y-axis represents the fraction of problems. The closer a curve is aligned to the Y-axis the superior is its performance relative to the other schemes over a range of 50 inputs.

Performance: Community Detection

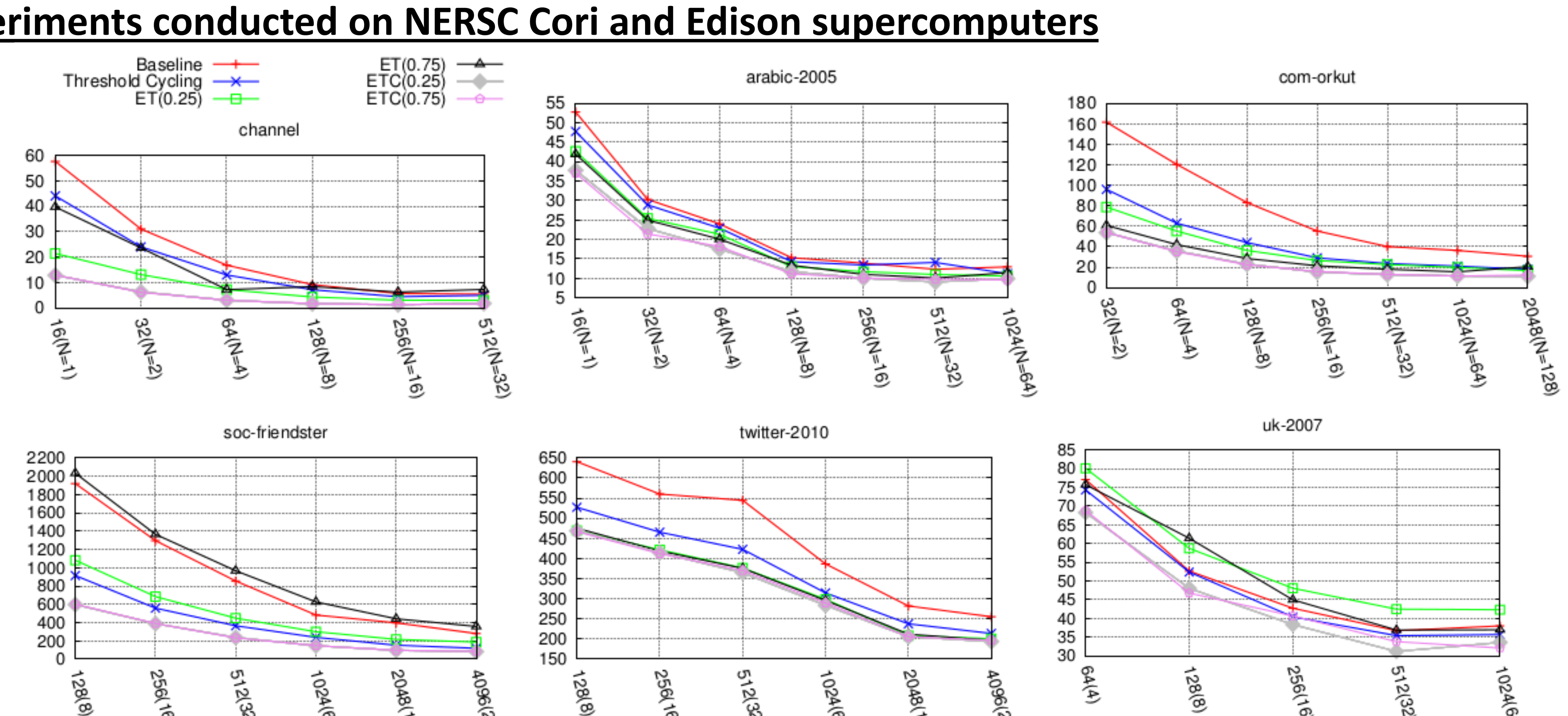
Experiments conducted on NERSC Cori and Edison supercomputers



Execution time   **Modularity**

Performance of Graph Challenge Input graphs on 16 nodes of Edison (except 200K case, which was run on 1 node).

Coloring performance is ~8-10x worse, modularity improves by an equal factor!



Observed 2-46x speedup relative to a parallel baseline version on real-world graphs!

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

Implemented communication intensive parts using MPI collectives (COLL), blocking Send-Recv (SR), nonblocking Send-Recv (NBSR) and RMA. Observed 4-18% divergence in performance across versions.

VERSIONS YIELDING THE BEST PERFORMANCE OVER THE BASELINE VERSION (RUN ON 16-128 PROCESSES) FOR INPUT GRAPHS (LISTED IN ASCENDING ORDER OF EDGES).

Graphs	Best speedup	Version
channel	46.18x	ETC(0.25)
com-orkut	14.6x	ETC(0.75)
soc-sinaweibo	3.4x	Threshold Cycling
twitter-2010	3.3x	ETC(0.25)
nlpkkt240	8.68x	Threshold Cycling
web-wiki-en-2013	7.92x	ET(0.75)
arabic-2005	5.8x	ETC(0.25)
webbase-2001	7x	ETC(0.25)
web-cc12-PayLevelDomain	3.75x	ETC(0.25)
soc-friendster	23x	ETC(0.25)
sk-2005	1.8x	ETC(0.75)
uk-2007	2.4x	ETC(0.75)

Energy/Memory for matching on Cori

Ver.	Mem. (MB/proc.)	Node eng. (kJ)	Node pwr. (kW)	Comp. %	MPI %	EDP
Friendster (1.8B edges)						
NSR	977.7	2868.04	10.7	61.6	38.4	8.29E+08
RMA	577.4	793.27	9.78	21.4	78.6	1.35E+08
NCL	419.3	740.13	9.65	20.8	79.1	1.27E+08
Stochastic block partition graph (475.1M edges)						
NSR	154.8	485.80	8.18	57.5	42.5	2.88E+07
RMA	196.3	690.41	9.09	7.2	92.8	5.24E+07
NCL	149	593.90	8.82	7.2	92.7	4.00E+07
HV15R (283.07M edges)						
NSR	210.2	154.98	5.95	13.5	86.4	4.04E+06
RMA	116.8	163.97	6.32	4.6	95.3	4.25E+06
NCL	106.9	140.85	6.07	3.2	96.7	3.27E+06

- Average memory consumption for NCL is the least, ~1.03 – 2.3x less than NSR, ~9–27% less than RMA
- Overall node energy consumption of NSR is about 4x to that of NCL and RMA for Friendster

References

- S. Ghosh, M. Halappanavar, A. Tumeo, A. Kalyanaraman, H. Lu, D. Chavarrià-Miranda, A. Khan, A. Gebremedhin "Distributed Louvain Algorithm for Graph Community Detection", *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*
- S. Ghosh, M. Halappanavar, A. Tumeo, A. Gebremedhin, "miniVite: A Graph Analytics Benchmarking Tool for Massively Parallel Systems", 2019 Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)
- S. Ghosh, M. Halappanavar, A. Tumeo, A. Kalyanaraman, A. Khan, A. Gebremedhin, "Exploring MPI Communication Models for Graph Applications Using Graph Matching as a Case Study" [under review]

Acknowledgements

The research is in part supported by the U.S. DOE ExaGraph project, a collaborative effort of U.S. DOE SC and NNSA at DOE PNNL.