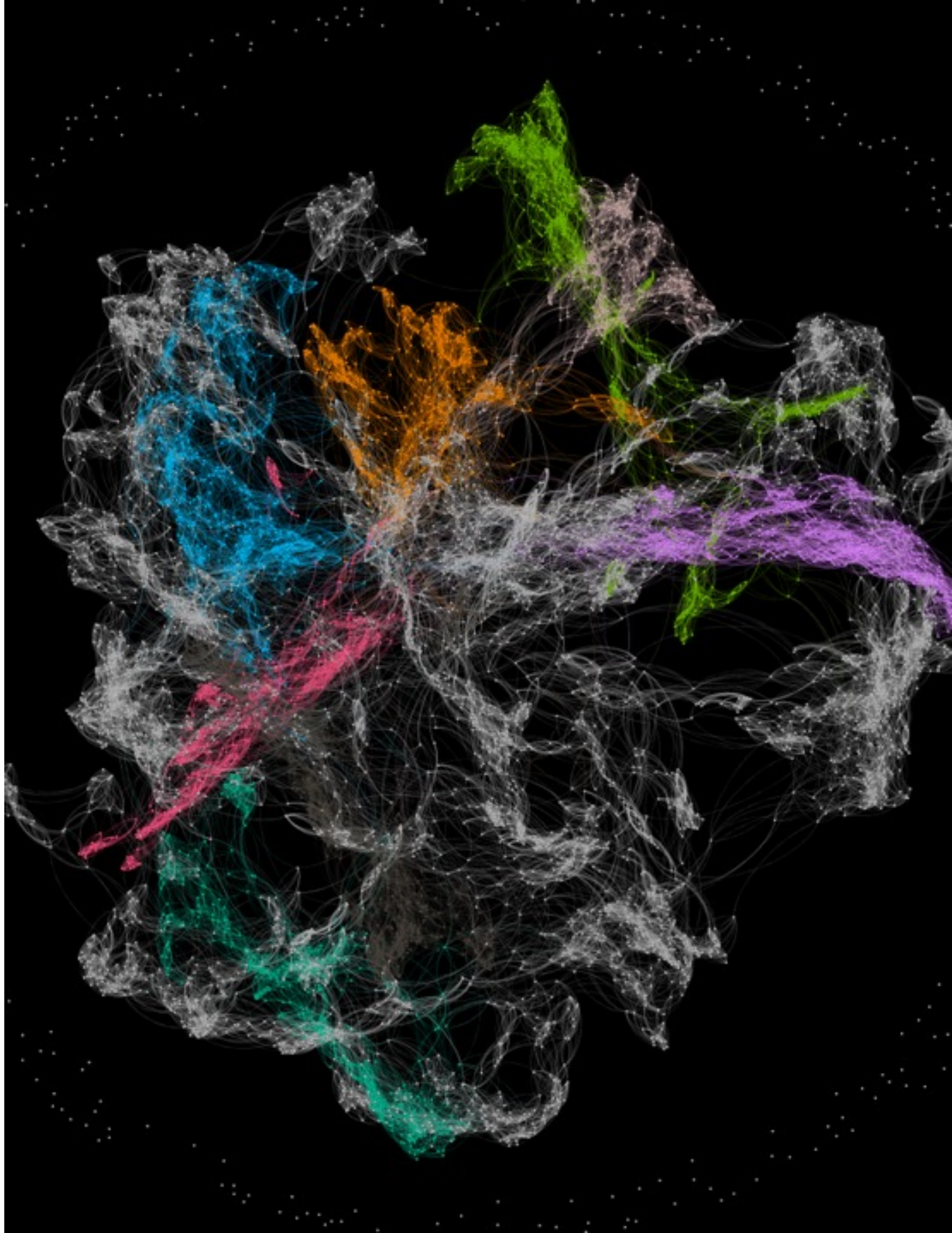
Pacific Northwest
NATIONAL LABORATORY

Using MPI RMA in Graph Analytics

Sayan Ghosh,

Computer Scientist

Physical and Computational Sciences Directorate,
Pacific Northwest National Laboratory

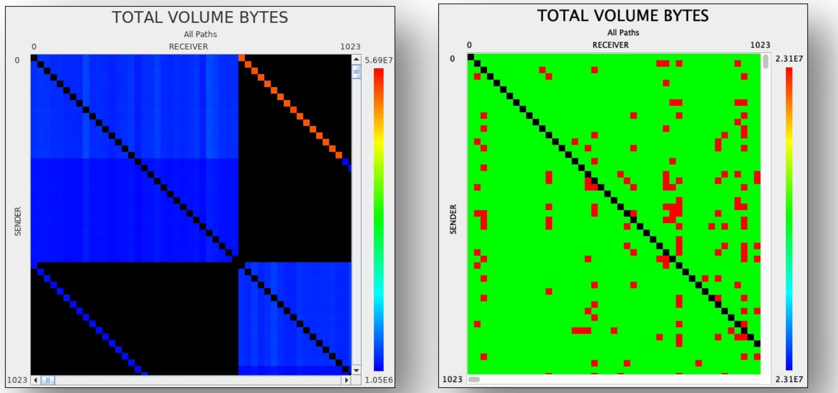


Graph algorithms

- Combinatorial (graph) algorithms are key enablers in data analytics
  - Graph coloring, matching, community detection, centralities, traversals, etc.
- Relatively *less* computation and *more* memory accesses
- Exact algorithms exorbitant – *heuristics* or *approximation* algorithms favored
- Graphs are multifarious, distributed-memory poses challenges
  - Asynchronous, irregular and adversarial communication patterns
  - Network contention can further exacerbate performance

```
Input: G(V,E), s ∈ V
Q.enqueue((visit(s))
while (!Q.empty()):
  u = Q.dequeue()
   for v in neighbor(u):
    if (!visited(v):
      Q.enqueue((visit(v))
```

```
Input: G(V,E)
for u in V:
  for v in neighbor(u):
    visit(v)
```



Pair-wise communication volume for BFS (left) and Graph neighborhood (right) for same graph

Communication is shown as a heat map of bytes/process pair, black is 0 bytes

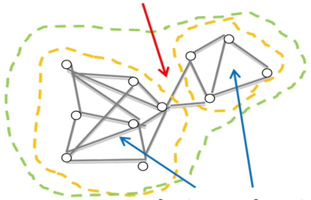
Graph500 or traversal-based algorithms are not necessarily representative use cases!

Ghosh, S., Tallent, N. R., and Halappanavar, M. 2021. Characterizing Performance of Graph Neighborhood Communication Patterns. *IEEE Transactions on Parallel and Distributed Systems*, *33*(4), 915-928.

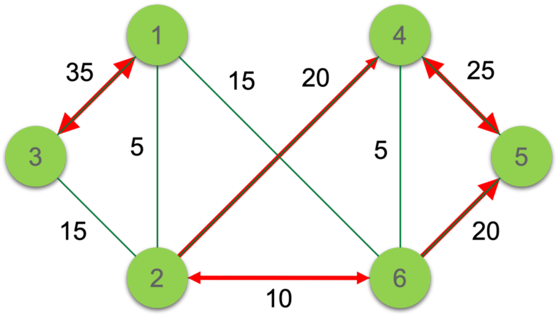
Graph applications: clustering and matching

Pacific Northwest
NATIONAL LABORATORY

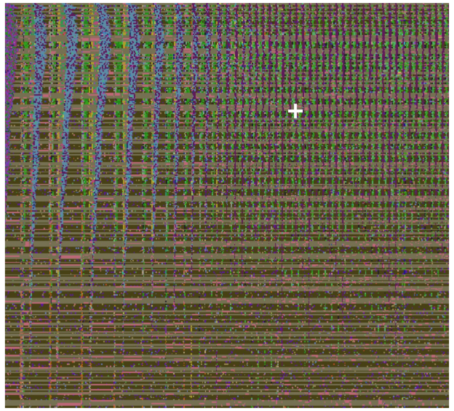
- Graph Clustering using Louvain
  - *Access pattern*: <u>Neighborhood search</u> and determine target community
  - *Synchronization*: Tasks require global knowledge, all-to-all pattern
  - <u>Latency</u> of message exchanges
- Graph Maximum Weighted Matching
  - *Access pattern*: <u>Neighborhood search</u> and determine heaviest active neighbor
  - *Synchronization*: Tasks are asynchronous, individual completion
  - <u>Bandwidth</u> (queue of messages)



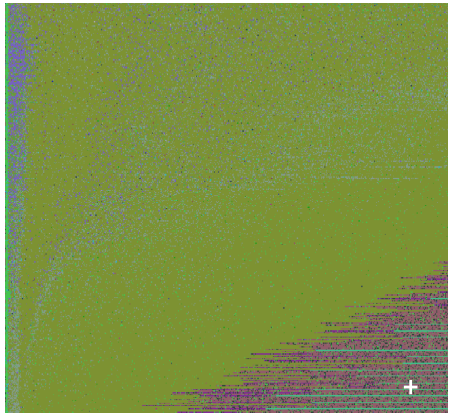
Goal: Identify tightly knit groups that strongly correlate to one another within their group, and sparsely so, outside.



Goal: Identify subset of edges s.t no two edges are incident on the same vertex.



Communication traces, *purple* denotes synchronization



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

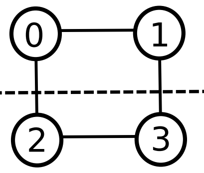
Ghosh S, Halappanavar M, Kalyanaraman A, Khan A, Gebremedhin A. Exploring MPI Communication Models for Graph Applications Using Graph Matching as a Case Study. In 2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS) 2019 May 20 (pp. 761-770). IEEE.

Codes: https://github.com/Exa-Graph

Communication pattern

- Asynchronous point-to-point updates
- MPI Send/Recv has been supporting messaging needs of parallel graph applications

Graph



Adjacency matrix

	0	1	2	3
0	0	1	1	0
1	1	0	0	1
2	1	0	0	1
3	0	1	1	0

Per-process CSR

Process #0
rowptr: 0 2 4
colidx: 1 2 0 3

Process #1
rowptr: 0 2 4
colidx: 0 3 1 2

Vertex-based graph distribution over two processes for undirected graph with 4 vertices and 8 edges. *Ghost vertices* for processes #0 and #1, respectively: 2 and 3; 0 and 1.

- Send/Recv matches well with the *owner computes* model

Input: $G_i = (V_i, E_i)$ portion of the graph G in rank i.

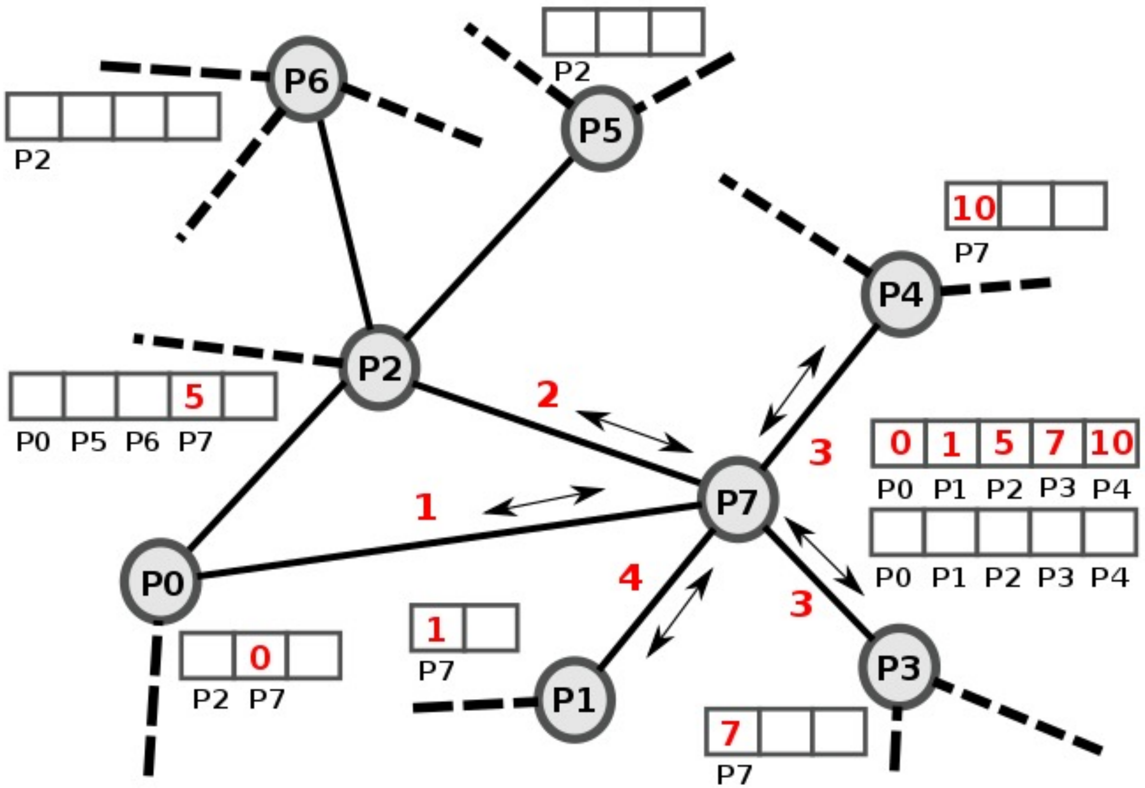
while true **do**
  $X_g \leftarrow$ **Recv** messages
  for $\{x, y\} \in X_g$ **do**
    $Compute(x, y)$ {local computation}
  for $v \in V_i$ **do**
    for $u \in Neighbor(v)$ **do**
      $Compute(u, v)$ {local computation}
      if $owner(u) \neq i$ **then**
        Nonblocking **Send**(u, v) to $owner(u)$
  if *processed all neighbors* **then**
    break and output data

(Lines 2–4: **2**; lines 5–9: **1**; lines 10–11: **3**)

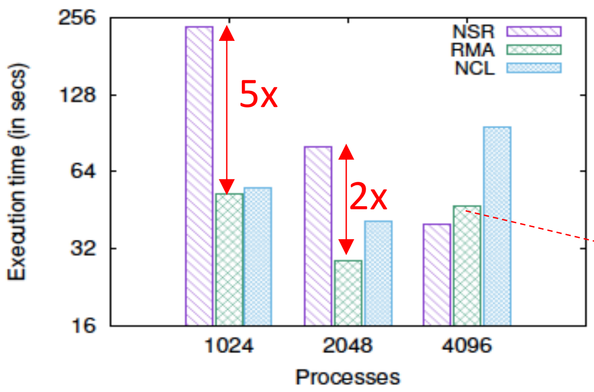
- However, MPI offers other models – better suited to *reduce synchronization* and *exploiting neighborhood communication* patterns

Calculating remote offset for RMA

- Trivial to calculate vertex *owner* in a *vertex-based* distribution
- However, RMA versions must calculate remote offset – *challenging for sparse data*
- Assuming passive target communication
- A process maintains two *O(#neighbor)* buffers (see P7)
  - One for storing prefix sums of (#ghosts – 1)
  - Second buffer obtained through `alltoall` exchanges of the above buffer among neighbors

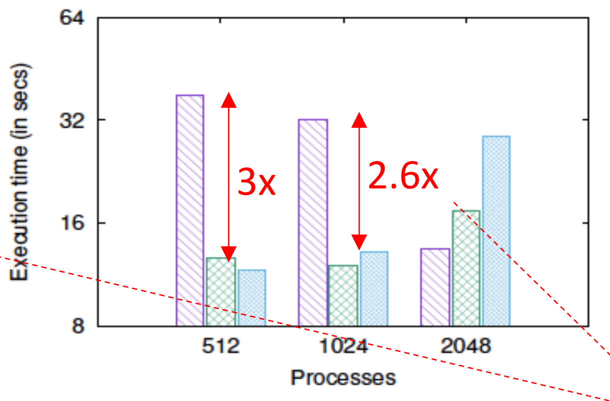


Performance of matching/clustering



(a) Friendster (1K-4K processes)

2K to 4K processes: E_p increases 4x



(b) Orkut (512-2K processes)

512 to 2K processes: E_p increases 14x

E_p Edges in process graph

d Degree stats (max/avg/stddev)

p	$\|E_p\|$	d_{max}	d_{avg}	σ_d
Friendster on 2K/4K processes				
2048	2.09E+06	2047	2045	2045.29
4096	8.33E+06	4095	4069	4069.87
Orkut on 512/2K processes				
512	1.30E+05	511	509	509.03
2048	1.84E+06	2047	1797	1808.03

NCL/RMA must communicate an extra data point (action), increasing communication overhead at scale

Graph Matching performance on NERSC Cori

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

Clustering is a nondeterministic problem

Dissimilar number of iterations across versions affect execution time and modularities (metric of quality)

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

MPI RMA alternatives: UPC++

- MPI-3 features such as Passive RMA and Neighborhood Collectives can be difficult to program or may have overheads
  - Status of proposal for RPC in MPI RMA ???
  - Neighborhood collective currently uses point-to-point internally (no h/w collectives)
- C++ enables performance portability models and modern applications – C++ interface of MPI is deprecated since MPI 2.2
- UPC++ has convenient one-sided, serialization/non-contiguous support and RPC interfaces; targets both performance and programmability

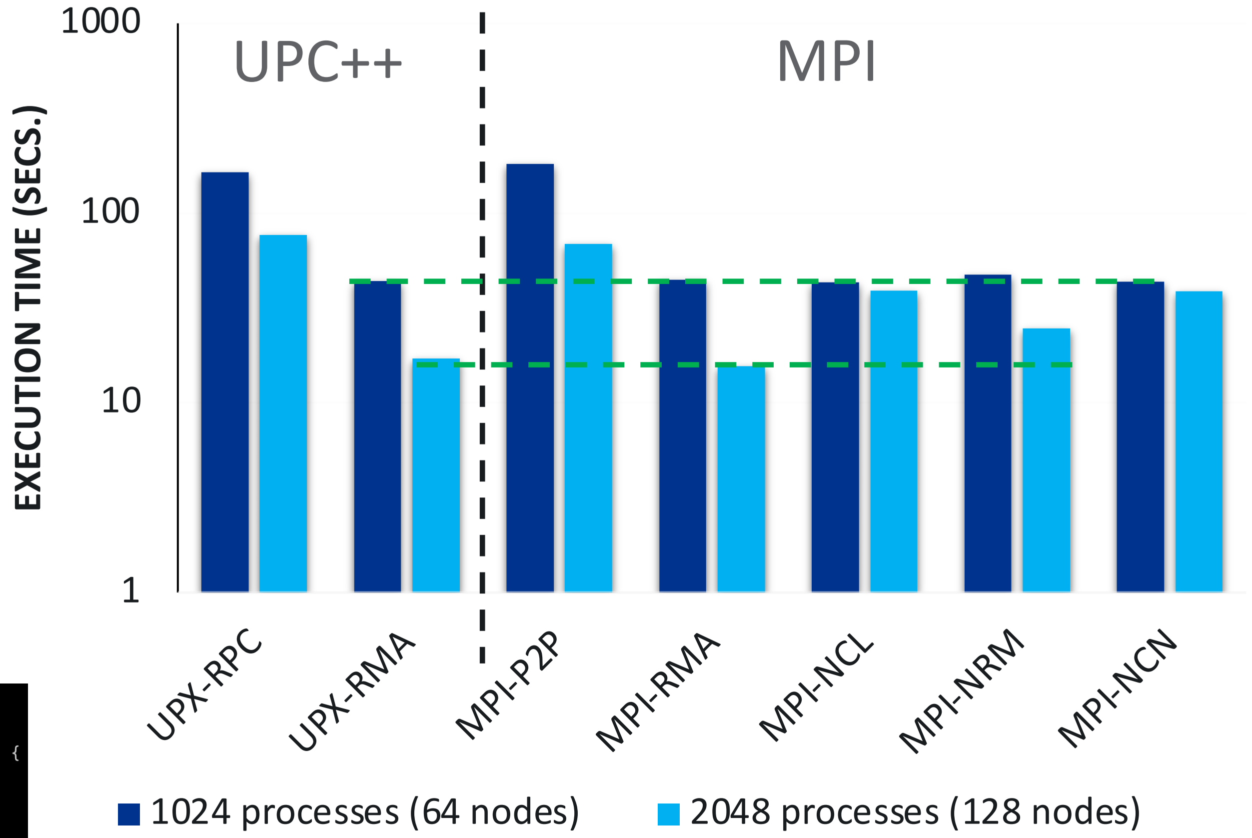
UPC++ Graph Matching performance

- UPC++-RMA performance competitive to MPI-RMA
- UPC++-RPC provides much better programmability (reduced ~100 LoC)
  - RPC provides mechanism to combine outgoing data with remote-side logic

```
GraphElem data_0 = data[0];
GraphElem data_1 = data[1];
current = upcxx::when_all(current,
        upcxx::rpc(target, [data_0, data_1](upcxx::dist_object<MaxEdgeMatchUPXRPC*>& dobj) {
            MaxEdgeMatchUPXRPC *here = *dobj;

            here->deactivate_edge(data_0, data_1);

            // recalculate mate[x]
            if (here->mate_[here->g_->global_to_local(data_0)] == data_1) {
                here->find_mate(data_0);
            }
            }, dobj));
```



Using cray-mpich/7.7.10 and upcxx/2020.3.8-snapshot on NERSC Cori (Haswell), input graph is com-Friendster (1.8B edges).

Acknowledgements

- Mahantesh Halappanavar, Nathan Tallent and Antonino Tumeo (PNNL)
  - Discussions on graph algorithms, implementations and performance analysis
- PAGODA team members, LBNL
  - For all the help and support with UPC++ version of Graph Matching
- PNNL Data-Model Convergence PACER LDRD project
- ECP ExaGraph Co-Design center

Thank you

UPC++ versions:

https://github.com/Exa-Graph/mel-upx

MPI versions:

https://github.com/Exa-Graph/mel

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