Simulating Application Agnostic Process Assignment for Graph Workloads on Dragonfly and Fat Tree topologies

Md Nahid Newaz,
Hua Ming
Oakland University, USA
{mdnahidnewaz,ming}@oakland.edu

Sayan Ghosh, Joshua Suetterlein,
Nathan R. Tallent
Pacific Northwest National Laboratory, USA
{sayan.ghosh,joshua.suetterlein,nathan.tallent}@pnnl.gov

Abstract

Distributed-memory graph applications are dominated by communication and synchronization overheads. For such applications, the communication pattern comprises of variable-sized data exchanges between process neighbors in a process graph topology, which unlike process grid for rectangular problems is difficult to optimize for enhancing the locality in a sustainable fashion.

Process assignment or remapping can improve the communication performance, however, existing solutions mostly caters to cartesian process topologies and not the graph topology. In this work, we propose application and topology agnostic process remapping strategies for graph applications. For two communication intensive distributed-memory graph applications (graph *clustering* and *triangle counting*), we demonstrate up to about 45% improvements in the overall MPI communication times through various remapping methodologies via SST-based packet-level simulations on Dragonfly and Fat Tree based network topologies and validate the strategies empirically on NERSC Perlmutter supercomputer.

1 Introduction

Distributed-memory graph workloads. Iterating over vertices and the associated edges of a graph in some order is a fundamental pattern in graph applications, one that leads to communication in the distributed-memory context because neighboring vertices can be owned by another process (which might be some network hops away from the current process)–edges whose endpoints are owned by different processes[1] are referred to as *cross edges*. This type of communication is often considered as an "adversarial" pattern due to the *incast* flows (multiple messages converging towards the same endpoint). Since real-world graphs are multifarious, relative process affinities transform as the graphs are distributed over varied number of processes, often leading to disparity in the communication. Even with an efficient graph partitioning, insufficient mapping/assignment can lead to arbitrary communication variability in modern network interconnects. The default strategy for process assignment to processor cores usually follows a shared-memory (SMP) style block placement, i.e., consecutively numbered MPI process ranks are placed sequentially on the same node. Fig. 1 captures the impact of standard SMP-style process placement, which may lead to an increased network traffic.

Existing work. Past approaches have either focused on process remapping for cartesian grids [2, 3], graph/mesh partitioning [1, 5, 9, 14] and process topology/neighborhood collectives optimizations [8, 11, 13, 15] in Message Passing Interface (MPI). Although the MPI topology creation routines provide an argument for remapping processes, most MPI implementations treat it as a no-op [8]. Performance tools such as CrayPAT [4] can automatically generate rank orders from application runs through introspection, but it only works consistently for regular cartesian domains.

Our approach. We posit that generating a rank-order is a one-time effort, which can be reused by various graph applications executing on the same process topology, even across platforms. To generate custom rank mappings, we access the process graph in certain order. We explore the consecutive vertex neighborhoods in the implicit process graph in parallel to generate the process reorder list (adjacent vertices are sorted in ascending order of their labels). This approach is scalable, because each MPI rank traverses its local process subgraph (only the processes it communicates with). Another variant considers the cross edges in the actual input graph between adjacent vertices in the process graph as *weights*, and, the vertex neighbors are traversed according to the order of their *weights*, as shown in Algorithm 1. The difference between the regular and weighted variants is the *order* that neighboring vertices (in the process graph) are listed for rank placement. We propose weighted variants (regular, ascending and descending) in this work, also comparing with random process placement.

[1]By *process* or *rank*, we refer to an *MPI process*.

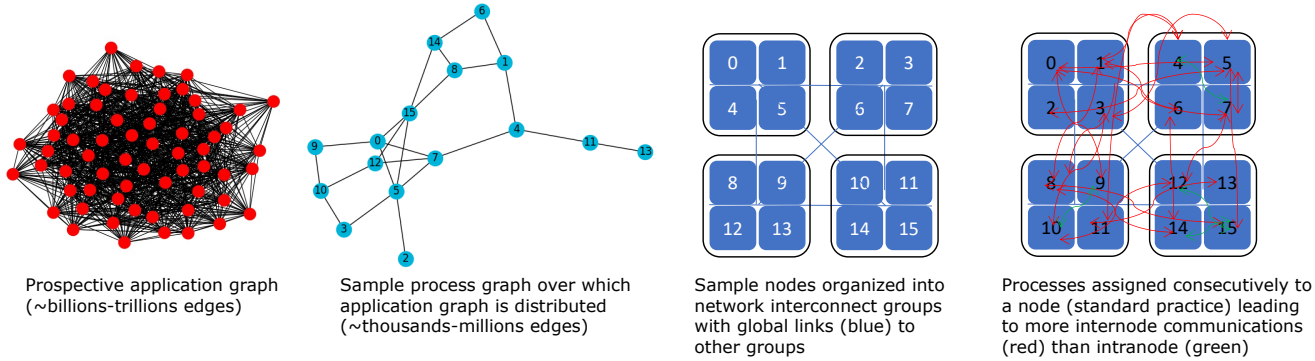


Figure 1: Importance of considering non-standard process assignment for graph workloads.

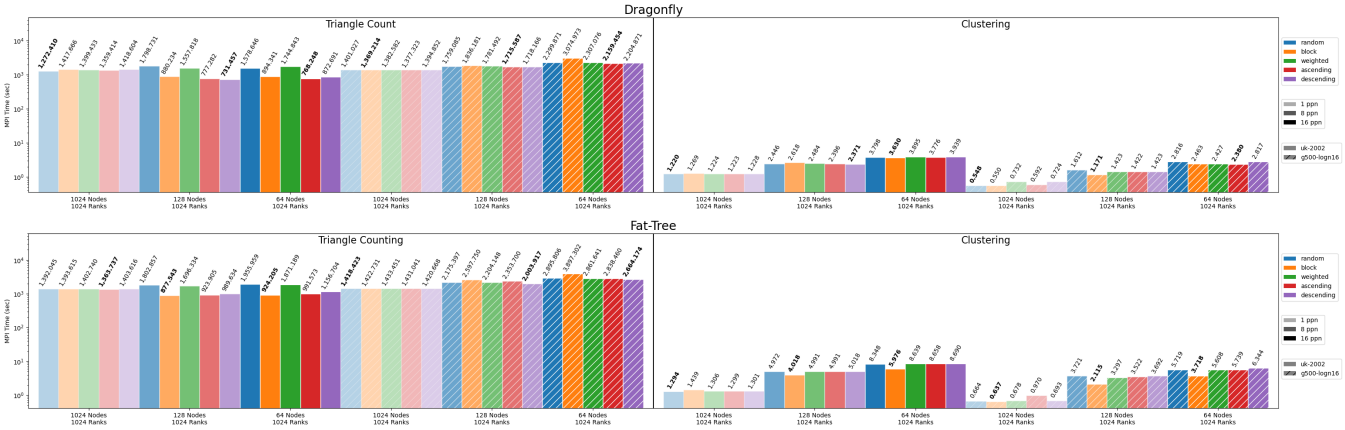


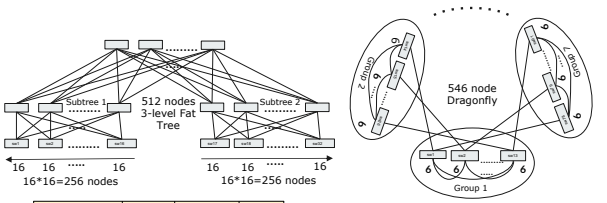
Figure 2: Simulation results of process remapping strategies on Dragonfly and Fat tree topology for two graphs on 1024 processes.

Algorithm 1 Generate process reordering based on the #cross edges.
Input: $G_i = (V_i, E_i)$, (undirected) graph G_i in rank i, *root* rank.
Output: R, reordered process list.

```
count ← {0} {optional, size of #processes}
for v ∈ V_i do
    for u ∈ adj(v) do {Neighbors of v}
        if owner(u) ≠ i then
            if owner(u) ∈ R ⇒ R ← R ∪ owner(u)
            count[owner(u)] += 1 {optional}
R ← sort(R, count) {optional}
R ← MPI_Gatherv(R, root = 0)
if rank == root then
    Remove duplicates in R
```

2 Evaluations

Simulation setup & platform. We used the packet-level parallel discrete event simulator Structural Simulation Toolkit SST-Macro [10] version 12.0; Dragonfly and Fat tree topology details are in Fig. 3. We simulate 512-node runs using multiple combinations of



Fat tree	#switches	switch radix	#nodes
(3,32,2): 3-level pruned w 2 subtrees, 16 leaf and aggregate switches/subtree	80	32	512
(3,48,2): 3-level pruned w 2 subtree, 24 leaf and aggregate switches/subtree	120	48	1152

Dragonfly	#switches	switch radix	#nodes
(13,7,6,6): 13 switches/group, 7 groups, 6 nodes/switch, 6 global ports/switch	91	24	546
(17,9,8,8): 17 switches/group, 9 groups, 8 nodes/switch, 8 global ports/switch	153	32	1224

Figure 3: Simulation setup for Dragonfly/Fat tree using SST-Macro.

processes-per-node (1, 4, 8 and 16), for up to hundreds of core hours on a four-way 2.5GHz Intel Xeon "Cascade Lake" CPU platform with 192GB memory and 40 CPU cores with 28MB shared L3 cache. We used GCC 10.2 and OpenMPI/4.1.1 for building the software codes. We confirm the simulation observations by evaluating on NERSC Perlmutter [16], using GCC 11.2 compiler (PrgEnv-gnu/8.3.3) and cray-mpich/8.2.25 (MPI). We use two communication-intensive graph applications: MPI-based graph clustering (miniVite [7]) and triangle counting (TriC [6]) with the recommended options, using input graphs from the SuiteSparse collection [12]. We use two graphs: `g500-logn16` (|E|=4.9M) and `uk-2002` (|E|=298M).

Graph Clustering (miniVite)

n(ppn)	uk-2002				g500-logn16			
	blk.	wgt.	asc.	dsc.	blk.	wgt.	asc.	dsc.
64(16)	4.81	6.09	3.49	3.36	0.59	1.05	0.75	0.85
128(8)	5.86	4.15	2.85	3.44	0.96	0.85	1.14	1.14

Triangle Counting (TriC)

n(ppn)	uk-2002				g500-logn16			
	blk.	wgt.	asc.	dsc.	blk.	wgt.	asc.	dsc.
64(16)	69.49	71.38	72.53	69.07	17.08	16.99	17.05	15.90
128(8)	69.42	67.81	72.82	68.91	16.39	16.36	17.12	16.49

Table 1: Avg. execution time (secs.) on 64/128 nodes of NERSC Perlmutter (Dragonfly topology) with distinct reorderings.

Observations. Our heuristics take a negligible amount of time (fraction of a second, depends on the process topology graph), since we use MPI for accessing the subgraphs and gathering the data from individual processes before writing it out to a file which is used for process assignment in SST-Macro. We also compare with the random rank mapping option in SST-Macro. Our results are shown in Fig. 2, and we show that in most cases, the default consecutive mapping is sub-optimal. Key takeaways are as follows:

- For smaller processes/node our process graph property based remapping can perform better across multiple applications and network topologies (Fig. 2 highlights the best variant).
- With relatively small-sized networks, increasing processes/node keeping the #processes fixed exploits only a small part of the overall topology which can be locally connected (within a group); in such cases our heuristics is not effective and the performance will depend on the application and input graph characteristics.
- Partitioning relatively small graphs on a large #processes can thwart the remapping heuristics (no specific irregular affinity characteristics, `alltoall` type pattern).
- Choice of the processes/node based on the input graph and network topological properties are critical for enhancing overall remapping performance.
- Empirical evaluations on NERSC Perlmutter (Table 1) confirms simulation results—up to 40% improvement observed relative to default block remapping in certain scenarios.
- Our results indicate that there is incentive for improving the current heuristics (e.g., considering separation of the high-degree nodes by strides) — the performance of random remapping also allude to possibilities for exploring better heuristics.

References

[1] Abhinav Bhatelé, Eric Bohm, and Laxmikant V Kalé. A case study of communication optimizations on 3d mesh interconnects. In *European Conference on Parallel Processing*, pages 1015–1028. Springer, 2009.

[2] Barbara Brandfass, Thomas Alrutz, and Thomas Gerhold. Rank reordering for mpi communication optimization. *Computers & Fluids*, 80:372–380, 2013.

[3] Daniel Chavarría-Miranda, Jarek Nieplocha, and Vinod Tipparaju. Topology-aware tile mapping for clusters of smps. In *Proceedings of the 3rd Conference on Computing Frontiers*, pages 383–392, 2006.

[4] Luiz DeRose, Bill Homer, Dean Johnson, Steve Kaufmann, and Heidi Poxon. Cray performance analysis tools. In *Tools for High Performance Computing*, pages 191–199. Springer, 2008.

[5] Karen Devine, Erik Boman, Robert Heaphy, Bruce Hendrickson, and Courtenay Vaughan. Zoltan data management services for parallel dynamic applications. *Computing in Science & Engineering*, 4(2):90–96, 2002.

[6] Sayan Ghosh and Mahantesh Halappanavar. Tric: Distributed-memory triangle counting by exploiting the graph structure. In *2020 IEEE high performance extreme computing conference (HPEC)*, pages 1–6. IEEE, 2020.

[7] Sayan Ghosh, Mahantesh Halappanavar, Antonino Tumeo, Ananth Kalyanaraman, and Assefaw H Gebremedhin. 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)*, pages 51–56. IEEE, 2018.

[8] William D Gropp. Using node information to implement mpi cartesian topologies. In *Proceedings of the 25th European MPI Users' Group Meeting*, pages 1–9, 2018.

[9] Bruce Hendrickson and Robert Leland. The chaco users guide. version 1.0. Technical report, Sandia National Labs., Albuquerque, NM (United States), 1993.

[10] Curtis L Janssen, Helgi Adalsteinsson, Scott Cranford, Joseph P Kenny, Ali Pinar, David A Evensky, and Jackson Mayo. A simulator for large-scale parallel computer architectures. *International Journal of Distributed Systems and Technologies (IJDST)*, 1(2):57–73, 2010.

[11] Christer Karlsson, Teresa Davies, and Zizhong Chen. Optimizing process-to-core mappings for application level multi-dimensional mpi communications. In *2012 IEEE International Conference on Cluster Computing*, pages 486–494. IEEE, 2012.

[12] Scott P Kolodziej, Mohsen Aznaveh, Matthew Bullock, Jarrett David, Timothy A Davis, Matthew Henderson, Yifan Hu, and Read Sandstrom. The suitesparse matrix collection website interface. *Journal of Open Source Software*, 4(35):1244, 2019.

[13] Seyed H Mirsadeghi, Jesper Larsson Traff, Pavan Balaji, and Ahmad Afsahi. Exploiting common neighborhoods to optimize mpi neighborhood collectives. In *2017 IEEE 24th international conference on high performance computing (HiPC)*, pages 348–357. IEEE, 2017.

[14] François Pellegrini and Jean Roman. Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. In *International Conference on High-Performance Computing and Networking*, pages 493–498. Springer, 1996.

[15] Mohammad Javad Rashti, Jonathan Green, Pavan Balaji, Ahmad Afsahi, and William Gropp. Multi-core and network aware mpi topology functions. In *European MPI Users' Group Meeting*, pages 50–60. Springer, 2011.

[16] Charlene Yang and Jack Deslippe. Accelerate science on perlmutter with nersc. *Bulletin of the American Physical Society*, 65, 2020.