An Integrated Approach for Accelerating Stochastic Block Partitioning

Frank Wanye
Dept. of Computer Science
Virginia Tech
Blacksburg, VA, USA
wanyef@vt.edu

Vitaliy Gleyzer
MIT Lincoln Laboratory
Lexington, MA, USA
vgleyzer@ll.mit.edu

Edward Kao
MIT Lincoln Laboratory
Lexington, MA, USA
edward.kao@ll.mit.edu

Wu-chun Feng
Dept. of Computer Science
Virginia Tech
Blacksburg, VA, USA
wfeng@vt.edu

***Abstract*—Community detection, or graph partitioning, is a fundamental problem in graph analytics with applications in a wide range of domains including bioinformatics, social media analysis, and anomaly detection. Stochastic block partitioning (SBP) is a community detection algorithm based on sequential Bayesian inference. SBP is highly accurate even on graphs with a complex community structure. However, it does *not* scale well to large real-world graphs that can contain upwards of a million vertices due to its sequential nature. Approximate methods that break computational dependencies improve the scalability of SBP via parallelization and data reduction. However, these relaxations can lead to low accuracy on graphs with complex community structure. In this paper, we introduce additional synchronization steps through vertex-level data batching to improve the accuracy of such methods. We then leverage batching to develop a high-performance parallel approach that improves the scalability of SBP while maintaining accuracy. Our approach is the first to integrate data reduction, shared-memory parallelization, and distributed computation, thus efficiently utilizing distributed computing resources to accelerate SBP. On a one-million vertex graph processed on 64 compute nodes with 128 cores each, our approach delivers a speedup of 322× over the sequential baseline and 6.8× over the distributed-only implementation. To the best of our knowledge, this Graph Challenge submission is the highest-performing SBP implementation to date and the first to process the one-million vertex graph using SBP.**

***Index Terms*—community detection, GraphChallenge, sampling, stochastic block partitioning, parallel computing, distributed computing**

I. Introduction

Community detection [1], also known as graph partitioning, is a fundamental problem in the field of graph analytics. It aims to find blocks of strongly connected vertices, such that vertices within a block are more strongly interconnected than vertices in different blocks. These communities tend to correspond to functional groups within a graph. Community detection has gained significant attention in a wide variety of domains including bioinformatics [2], [3], workload balancing [4], and recommendation systems [5].

Optimal community detection is an NP-hard problem, hence it is generally solved via heuristics. Of these heuristics, modularity maximization is the most common, with the Louvain [6] and Leiden [7] algorithms in particular gaining a lot of traction for their speed and relative accuracy. Much of the work on parallel and distributed community detection has centered around these methods, allowing them to scale to web-scale graphs with billions of edges [8]. However, the modularity maximization heuristic has some well-documented shortcomings. The most well-known of these is the resolution limit [9], which affects the algorithm's performance when the community sizes are highly variable. Other descriptive community detection algorithms demonstrate similar shortcomings, while others still require prior knowledge of the number of communities [10], which is often unavailable in real-world scenarios.

Stochastic block partitioning (SBP) [11], [12] is a community detection method belonging to the inferential class of community detection algorithms [13]. These algorithms are based on inference methods and are not prone to overfitting. As such, they provide higher quality community detection results than the descriptive class of community detection methods [14] However, these methods are considerably slower than their descriptive counterparts. This limits their applicability to large, real-world graphs, which often have upwards of a million vertices and edges. This work aims to improve the practical usability of SBP, thus enabling accurate community detection at scale.

Various approximate methods have previously been used to accelerate SBP, including data reduction [15]–[17], parallelizing the Markov chain Monte Carlo (MCMC) phase of SBP [18], [19], and distributed computation using incomplete data [19], [20]. However, these methods on their own do not provide enough speedup to scale SBP to very large graphs. Moreover, they run the risk of poor convergence, especially when the graph structure is complex, which could

This material is based upon work supported by the Under Secretary of Defense for Research and Engineering under Air Force Contract No. FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Under Secretary of Defense for Research and Engineering.

This work was supported in part by NSF I/UCRC CNS-1822080 via the NSF Center for Space, High-performance, and Resilient Computing (SHREC).

be exacerbated when the methods are used together. To solve this problem, we first introduce a vertex-level batching scheme to improve the accuracy of parallel and distributed SBP approximations via additional synchronization. We then leverage batching to develop an approximate SBP implementation that integrates data reduction, parallel MCMC, and distributed computation. This integrated SBP approach scales to large graphs while maintaining the high accuracy of sequential SBP - it processes the 1 million vertex graph in under 12 minutes without sacrificing accuracy. To the best of our knowledge, this Graph Challenge submission is the highest-performing SBP implementation to date and the first to successfully process a graph of that size using SBP.

Our main research contributions are as follows:

- A vertex-level data batching scheme that improves the accuracy of existing parallel and distributed SBP approximations on complex graphs from normalized mutual information (NMI) scores of ≤ 0.63 (and as low as 0.0) to ≥ 0.87 (see Table III).
- An integrated high-performance implementation of SBP that combines and optimizes several state-of-the-art methods into one unified approach.
- A rigorous evaluation of our approach on 64 compute nodes that demonstrates speedups as high as $322\times$ over single-node SBP running sequential MCMC and up to $6.8\times$ over distributed computing alone while maintaining accuracy.

II. Background

In this section, we provide background information about SBP and various acceleration techniques that are enabled via algorithmic relaxations of sequential SBP.

A. Stochastic Block Partitioning (SBP)

SBP [11], [12] is an inferential community detection algorithm [13] based on the degree-corrected stochastic blockmodel (DCSBM) [21]. The DCSBM is a representation of graph structure based on community (block) connectivity. It is stored as a matrix M, where, in a directed graph, $M_{a,b}$ contains the number of edges that originate from vertices in community a and have vertices in community b as their destination.

SBP infers the best vertex-to-community assignment, as well as the optimal number of communities, by minimizing the description length H of the DCSBM, given by the following equation:

$$H = Eh\left(\frac{B^2}{E}\right) + V\log B - \sum_{r,c}\log\frac{M_{r,c}}{d_{r,out}d_{c,in}},$$

where E, V, and B are the number of edges, vertices, and blocks in the graph, $d_{a,out}$ and $d_{a,in}$ are the out- and in-degree of block a, and $h(x) = (1+x)\log(1+x) - x\log x$ [12], [22].

The inference itself is performed iteratively, in two alternating phases - the block merge phase, where entire blocks are merged based on the resulting change in H, and the Markov chain Monte Carlo (MCMC) phase, where individual vertices move between blocks via the Metropolis-Hastings algorithm. The MCMC phase is the main driver of the inference, while the block-merge phase prevents the algorithm from getting stuck in a local minimum and allows for the use of a Fibonacci search to find the optimal number of communities.

B. Sampling from Graphs

There has been a significant amount of work into using sampling to reduce the storage and computational demands of processing large graphs [15], [23]–[26]. As such, several sampling algorithms have been developed over the years. However, as demonstrated in [24], the optimal choice of sampling algorithm strongly depends on the specific processing task. This is because graphs have many directly and indirectly measurable parameters, and devising a sampling algorithm that preserves all, or even most of them is not trivial.

Community structure is an indirectly measurable graph parameter. The expansion snowball [17], [26] and maximum degree [17] sampling algorithms have been shown to preserve community structure on sparse graphs. A previous Graph Challenge entry [15] leveraged uniform random sampling to accelerate SBP while maintaining accuracy. These works also show that the community structure inferred from the sampled subgraph can be extended to the entire graph. Additionally, degree-preserving graph summarization using supernodes has been shown to preserve community structure [16].

C. Parallelizing the MCMC Phase

The block merge phase of the SBP algorithm is embarrassingly parallel and thus trivial to parallelize with a multithreaded framework such as OpenMP. However, the MCMC phase of SBP is inherently sequential [27] and thus difficult to parallelize without the potential for losing inference accuracy. In [19], a batch-based shared memory approach with a single master thread determining update frequency was employed to parallelize SBP within one node.

Another shared memory parallel SBP approach is the Hybrid SBP algorithm [18]. In [27], it was shown that asynchronous Gibbs sampling, an embarrassingly parallel method, can be used to parallelize MCMC so long as the conditional dependency distribution of the underlying model is sparse enough. However, when this condition is false, asynchronous Gibbs does not converge well, and finding out whether or not this condition is true is not trivial. Hence, Hybrid SBP takes advantage of this by processing a small number of highly influential, high-degree vertices sequentially using Metropolis-Hastings and the remaining majority of low-degree vertices via an adapted asynchronous Gibbs algorithm.

D. Distributed Computing

Distributing the SBP algorithm is a difficult task. In addition to the inherently sequential nature of the MCMC phase, the underlying blockmodel needs to be accessed both row- and column-wise, further complicating the problem of data distribution. A divide-and-conquer approach [19] was proposed that

partitions the graph into separate subgraphs and runs shared-memory parallel SBP independently on each subgraph, before combining and then fine-tuning the partial results. However, processing the subgraphs independently breaks many computational dependencies in the MCMC stage. Thus, on more than 8 nodes, accuracy degrades significantly.

An alternative distributed SBP algorithm, EDiSt [20], tackles this problem by duplicating the graph and blockmodel data across all compute nodes. By introducing internode communication across compute nodes, which are each tasked with processing a distinct set of vertices, the degree to which computational dependencies are broken is greatly reduced. This allows EDiSt to scale up to a much larger number of nodes without accuracy degradation, at the cost of increased memory requirements.

The three methods described above have for the most part been used independently, with no single approach combining all three, as seen in Table I.

TABLE I
SUMMARY OF PRIOR WORK

Reference	Data Reduction	Shared-Memory Parallelization	Distributed Computation
[15]	YES	-	-
[17]	YES	-	-
[16]	YES	-	-
[19]	-	YES	YES
[18]	-	YES	-
[20]	-	YES	YES
This manuscript	YES	YES	YES

III. METHODOLOGY

In this section, we describe our integrated SBP framework and the refinements made to the SBP approximations that make up the framework.

A. The Integrated SBP Framework

The integrated SBP framework consists of three approximate SBP algorithms that perform data reduction, parallel MCMC computations, and distributed computation using incomplete data.

Sampling: We select the SamBaS [17] framework as our data reduction approach. It has been shown to work well on both real-world and synthetic graph datasets. SamBaS consists of four steps:

1) Sampling to produce a sampled subgraph with a fraction of the full graph's vertices.
2) Community detection (via SBP) on the sampled subgraph.
3) Extension of the community memberships obtained from the sampled subgraph to the full graph.
4) Finetuning the results using the Metropolis-Hastings algorithm.

This four-step approach allows for relatively easy substitution of the single-threaded SBP algorithm with any other accelerated variant.

For the sampling step, we implement three different sampling algorithms: uniform random sampling, expansion snowball sampling, and maximum degree sampling. In prior work [15], [17], these algorithms have been shown to perform well on a variety of graphs.

Shared-Memory Parallelization of MCMC: We implement the hybrid SBP [18] algorithm to parallelize the MCMC phase of SBP. This method has been shown to maintain accuracy on a wide array of graphs while significantly speeding up the MCMC phase. Moreover, it processes most of the vertices within each iteration asynchronously, which potentially reduces the amount of inter-node communication needed when this method is distributed. When integrating hybrid SBP with SamBaS, we replace the Metropolis-Hastings algorithm in Steps 2 and 4 with the hybrid parallel MCMC algorithm.

Distributed Computation: Finally, we implement EDiSt [20] as our distributed computation algorithm, because of its ability to scale to a large number of compute nodes without losing accuracy. This increase in scalability enables larger speedups than those observed with the divide-and-conquer approach [19]. Due to the low frequency of inter-node communication, each MPI rank in EDiSt operates on an incomplete blockmodel with stale updates.

When integrating hybrid SBP with EDiSt, we run a single iteration of the hybrid SBP MCMC phase independently within each MPI rank. Then, we use MPI all-to-all communication to synchronize vertex moves across all MPI ranks.

When integrating EDiSt with SamBaS, much like in the case of hybrid SBP, we replace the SBP algorithm in the community detection step with EDiSt. Furthermore, we take advantage of the distributed MCMC phase from EDiSt to perform the finetuning step of SamBaS in a distributed fashion.

To ensure consistency across MPI ranks, the SamBaS sampling and extension steps are run single-threaded and on a single MPI rank. The results are then communicated to the other MPI ranks using MPI `broadcast` operations. While it is possible to parallelize at least some of the code in these steps, we find that the single-threaded execution of these 2 phases does not take a significant amount of time for all graphs considered in our evaluation.

The complete integrated approach is illustrated in Figure 1.

B. Additional Refinements

The Python baseline for SBP is prohibitively slow, requiring over 2 hours of computation to process the official 50,000 vertex graphs. On the other hand, the C++ baseline from the `graph-tool` software is much faster, but difficult to modify due to being part of a large and complex library. To speed up the computation, as well as make it easier to apply optimizations to our code, we translate the Python baseline code into C++. This translation allows us to take advantage of compile-time optimizations, efficient multithreading with OpenMP, and highly performant data structure libraries like `tessil`. We then implement several optimizations to improve the runtime of our C++ baseline code. Additionally, we implement a vertex-level batching scheme to improve the accuracy

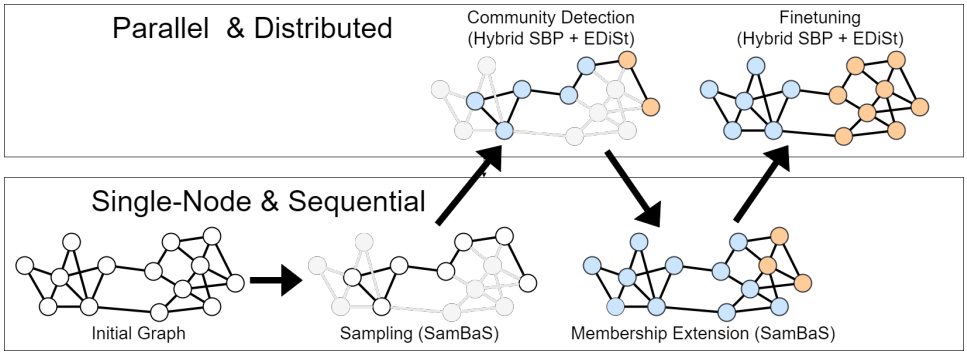

Fig. 1. An illustration of the integrated approach to accelerating SBP.

of parallel and distributed SBP approximations on complex Graph Challenge graphs.

Optimizations: We outline the most effective optimizations we implement in our C++ SBP code below:

- storing both the blockmodel matrix *and* its transpose, allowing efficient row-wise and column-wise iteration at the cost of having to update two matrices whenever the blockmodel changes. Because the number of accesses is much higher than the number of accepted moves, this results in significant time savings.
- storing each blockmodel matrix as a vector of `tessil robin_map` objects, which allow fast iteration *and* modification of the blockmodel matrix, while also saving memory through representing the blockmodel in sparse matrix format. When put together with transpose matrix storage, this becomes similar to the compressed representation optimization described in [28].
- implementing a disjoint-set data structure to store block merges. Previously, to merge block a into block b, an $O(V)$ lookup would have to be performed to find all vertices belonging to block a, before updating their block membership. This means that performing x merges takes $O(xV)$ time. The disjoint-set data structure allows all block memberships to be updated in a single iteration, in approximately $O(V)$ time.
- computing the change in description length δH using a vector of changes to the blockmodel matrix Δ. This removes the need for storing entire rows and columns of the blockmodel to calculate δH and allows for more efficient updates to the blockmodel matrix since only matrix cells that are changed are accessed.
- precomputing a cache of `log` values to reduce time spent computing logarithms for H and δH.
- parallelizing several trivial but computationally expensive portions of the code using OpenMP, including the δH computations in the block merge phase, building the blockmodel matrix, and sorting vertices.

Improving Accuracy: Both the parallel and distributed SBP implementations described in III-A are approximations of Metropolis-Hastings that rely on the assumption that the input graph has a power-law degree distribution, with many low-degree vertices and few high-degree vertices. The Graph Challenge graphs, however, have a truncated degree distribution, with almost no 1 and 2-degree vertices and no vertices with degrees higher than 100. Coupled with the low frequency of synchronization in hybrid SBP and EDiSt, this means that both methods occasionally lead to inaccurate community detection on complex graphs. We introduce vertex-level batching to enhance the accuracy of these methods.

The batching approach splits vertices into equal-sized disjoint sets. The batches are then processed one at a time, with communication occurring at the end of each batch to ensure consistency of the blockmodel across all computing elements. With the parallel implementation (hybrid SBP), the few highly influential vertices are already processed sequentially, so no additional synchronization is needed for those vertices. Thus, we only process the low-influence vertices using the batched approach. At the end of each batch, a single thread updates the blockmodel matrix using the accepted vertex moves.

In the parallel + distributed implementation, an MPI all-to-all communication call is used to synchronize accepted vertex moves across all MPI ranks. Additionally, we add a synchronization step after the highly influential vertices are processed. Finally, we also include batching in the distributed implementation with sequential MCMC, except in this case all vertices are split into batches.

In addition to this, we find that the parallel MCMC implementation leads to better quality partitions with a more accurate approximation of vertex influence based on the information content of edges, as described in [29]. Instead of selecting highly influential vertices based solely on their degree, we sort all edges in the graph according to the degree product of their associated vertices. Then, we select vertices from the edges with the highest degree products.

With these refinements, we find that both the parallel and distributed implementations converge and match baseline accuracy using just 2 batches (that is, just one additional synchronization step). While this does typically negatively impact runtime, in some cases the improvement in convergence negates the extra communication overhead.

IV. Experiments

Here we describe our experimental approach and the metrics used to evaluate it, followed by a description of the datasets and infrastructure on which we perform our experiments.

A. Experimental Approach

We evaluate the integrated approach by conducting an integration study over its three components (sampling, parallel MCMC, and distributed computing) to better understand the impact of each individual part, as well as their permutations. For sampling-based approaches, we run each of the three implemented sampling algorithms at the 50% sample size. For distributed computing-based approaches, we run 4 MPI ranks per node, as described in [20]. Finally, because SBP is nondeterministic, we perform each run on each graph 2 times and present the result with the lowest H.

B. Evaluation Metrics

We employ information theoretic metrics based on Shannon entropy to evaluate accuracy. As outlined in [12], precision and recall in the information theoretic domain are given by $\frac{I(T;O)}{I(O)}$ and $\frac{I(T;O)}{I(T)}$ respectively, where $I(T;O)$ is the mutual information between the true community assignments T and the algorithm's output O, $I(T)$ is the entropy of the true community assignments and $I(O)$ is the entropy of the algorithm's assignments. Due to space constraints, instead of reporting both metrics, we instead report the normalized mutual information (NMI), which is given by $\frac{I(T;O)}{\sqrt{I(T)\times I(O)}}$ [30].

To evaluate the runtime benefits of our integrated approach, we measure the speedup obtained, calculated via the equation: $\frac{\texttt{runtime of baseline}}{\texttt{runtime of accelerated SBP}}$. To ensure a fair comparison, we use our optimized C++ code without sampling, parallel MCMC, or distributed computation as the baseline.

Finally, our integrated approach sacrifices memory usage for increased runtime in several areas, including storing two sparse blockmodel matrices and duplicating data to enable distributed computing. To study our memory usage and better understand the practical limitations of this approach, we measure the maximum resident set size (RSS) of each run via the slurm `sacct` command. The maximum RSS value records the maximum amount of physical memory allocated to a process.

C. Datasets

To stress our approach, we restrict our experiments to the most complex Graph Challenge graphs with high block overlap and high block size. Table II summarizes the selected datasets.

TABLE II
Datasets Used for Evaluation

Number of vertices	Number of edges	Number of communities
$5,000$	$51,157$	19
$50,000$	$1,187,682$	44
$200,000$	$4,754,406$	71
$1,000,000$	$23,772,977$	125

D. Hardware

All our experiments were performed on the Virginia Tech Tinkercliffs cluster. The cluster is made up of 308 compute nodes equipped with 128-core AMD EPYC 7702 CPUs with 256GB of memory. The interconnect employed by the cluster is HDR-100 IB. Due to slurm job configuration limitations on Tinkercliffs, we are restricted to just 64 of the 308 available compute nodes for any given run. Thus, all our runs involving distributed computation are run on 64 compute nodes.

V. Results

In this section, we discuss our integration study results. Due to space limitations, where appropriate we only display results with the uniform random sampling algorithm, which we found to be an effective compromise between runtime and accuracy.

First, in Table III, we show 7 instances that highlight our work to improve convergence in parallel and distributed SBP via batching. In 4 of these instances, the approach completely failed to converge without batching, leading to uninformative partitions and an NMI score of 0.0. With batching, we achieve NMI scores of at least 0.87 in all 7 instances. Thus, all proceeding parallel and distributed results shown are run with additional synchronization using 2 batches.

TABLE III
Selected Results With and Without Additional Synchronization

Implementation	Graph size	NMI without batching	NMI with batching
Parallel	1000000	0.00	0.90
Distributed (Sequential)	1000000	0.00	0.88
Parallel + Sampling	200000	0.63	0.87
Parallel + Distributed	1000000	0.00	0.90
Sampling + Distributed (Sequential)	200000	0.47	0.87
Integrated	200000	0.00	0.88

As seen in Figure V (top), all approaches generally achieve similar NMI to the baseline, showing that there is little-to-no difference in the quality of results obtained with the integrated framework. The only exception is the 5000 vertex graph, whose NMI is comparatively low in all methods that rely on sampling. This may be due to the lower ratio of vertices-to-communities in this graph, making it easier to undersample communities. Sampling can also increase the variation in result quality. Thus, performing more runs could eventually lead to higher NMIs for the 5000 vertex graph.

The speedup results, summarized in Figure V (middle), show that combining different acceleration methods leads to much higher speedups. On the 1 million vertex graph, the integrated approach was 322× faster than the sequential baseline, reducing the runtime from over 63 hours to under 12 minutes. This is 6.8× faster than the 48× speedup obtained with distributed computation on its own. On smaller graphs, however, the overhead of the combined acceleration methods may occasionally lead to a lower speedup. Note that regardless of implementation, the speedups obtained increase with the

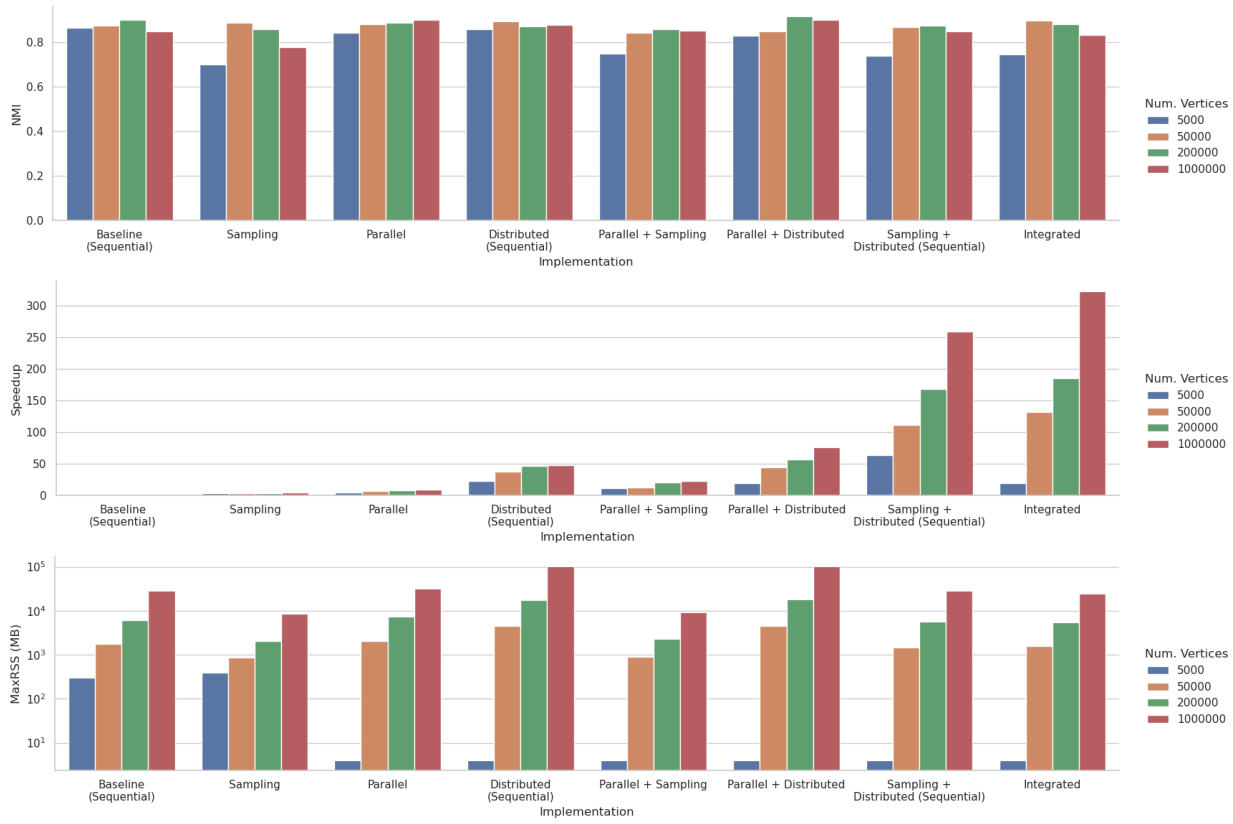

Fig. 2. NMI (top, higher is better), speedup over sequential baseline SBP (middle, higher is better), and memory utilization (bottom, lower is better) results from the integration study, run on 64 compute nodes with 128 cores each.

size of the graph as the SBP computation time increases faster than the overhead of accelerating it.

Finally, the memory utilization results are summarized in Figure V (bottom). Our chosen distributed SBP implementation duplicates the data across all compute nodes. This leads to significantly higher total memory utilization, up to 100GB on the 1 million vertex graph. The increased memory demand means that the size of the graphs that can be processed with this distributed approach is strongly limited by the amount of available memory. The inclusion of sampling in the integrated approach decreases the memory requirements by a quarter, to about 25GB, allowing larger graphs to be processed on the same hardware.

VI. Conclusion

In this manuscript, we present a scalable and integrated approach to accelerating stochastic block partitioning (SBP). To the best of our knowledge, this approach is the first to combine sampling, parallel MCMC computation, and distributed computing. The sequential SBP approximations on which these three methods are based sometimes suffer from reduced accuracy when applied to complex graphs. To improve the accuracy of these methods on such graphs, we enhance them with additional synchronization via vertex-level batching. This enhancement enables the parallel and distributed SBP approximations to achieve NMI scores of 0.87 or higher on graphs where they would otherwise achieve NMI scores of 0.63 or lower.

We evaluate our approach through an extensive integration study, demonstrating that on 64 nodes with 128 cores each, the integrated SBP approximation accelerates SBP by up to 322× over the sequential baseline without significantly affecting its accuracy. This reduces the processing time on the 1 million vertex graph from over 63 hours to under 12 minutes. Furthermore, the integrated approach uses about 75% less memory and achieves a speedup of up to 6.8× over distributed computation alone. In this manner, the integrated approach enables the processing of larger graphs with SBP than would otherwise be possible.

In future work, we aim to conduct comparative analyses of the runtime and accuracy of our integrated approach against widely used community detection algorithms like Louvain and Leiden. We also plan to scale SBP to real-world web-scale graphs with billions of edges by exploring data partitioning schemes for distributed SBP and further algorithmic refinements tailored specifically to SBP.

Acknowledgment

The authors acknowledge Advanced Research Computing at Virginia Tech for providing computational resources that have contributed to the results reported within this paper. URL: https://arc.vt.edu/

REFERENCES

[1] S. Fortunato, "Community detection in graphs," *Physics Reports*, vol. 486, no. 3-5, pp. 75–174, 2 2010. [Online]. Available: doi.org/10.1016/j.physrep.2009.11.002

[2] V. Oles, S. Dash, and R. Anandakrishnan, "BiGPICC: a graph-based approach to identifying carcinogenic gene combinations from mutation data," *bioRxiv*, p. 2023.02.06.527327, 2 2023. [Online]. Available: doi.org/10.1101/2023.02.06.527327

[3] J. B. Pereira-Leal, A. J. Enright, and C. A. Ouzounis, "Detection of functional modules from protein interaction networks," *Proteins: Structure, Function, and Bioinformatics*, vol. 54, no. 1, pp. 49–57, 9 2003. [Online]. Available: doi.wiley.com/10.1002/prot.10505

[4] G. M. Levchuk and J. Colonna-Romano, "Optimizing collaborative computations for scalable distributed inference in large graphs," in *Signal Processing, Sensor/Information Fusion, and Target Recognition XXVII*, I. Kadar, Ed., vol. 10646. SPIE, 2018, p. 23. [Online]. Available: doi.org/10.1117/12.2305872

[5] P. Krishna Reddy, M. Kitsuregawa, P. Sreekanth, and S. Srinivasa Rao, "A Graph Based Approach to Extract a Neighborhood Customer Community for Collaborative Filtering," in *Databases in Networked Information Systems*. Springer, Berlin, Heidelberg, 2002, pp. 188–200. [Online]. Available: doi.org/10.1007/3-540-36233-9_15

[6] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, p. P10008, 10 2008. [Online]. Available: doi.org/10.1088/1742-5468/2008/10/P10008

[7] V. A. Traag, L. Waltman, and N. J. van Eck, "From Louvain to Leiden: guaranteeing well-connected communities," *Scientific Reports 2019 9:1*, vol. 9, no. 1, pp. 1–12, 3 2019. [Online]. Available: doi.org/10.1038/S41598-019-41695-Z

[8] X. Que, F. Checconi, F. Petrini, and J. A. Gunnels, "Scalable Community Detection with the Louvain Algorithm," in *2015 IEEE International Parallel and Distributed Processing Symposium*. IEEE, 5 2015, pp. 28–37. [Online]. Available: doi.org/10.1109/IPDPS.2015.59

[9] S. Fortunato and M. Barthelemy, "Resolution limit in community detection," *Proceedings of the National Academy of Sciences*, vol. 104, no. 1, pp. 36–41, 1 2007. [Online]. Available: doi.org/10.1073/pnas.0605965104

[10] A. Lancichinetti and S. Fortunato, "Community detection algorithms: A comparative analysis," *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, vol. 80, no. 5, p. 056117, 11 2009. [Online]. Available: doi.org/10.1103/PHYSREVE.80.056117

[11] T. P. Peixoto, "Parsimonious Module Inference in Large Networks," *Physical Review Letters*, vol. 110, no. 14, p. 148701, 4 2013. [Online]. Available: doi.org/10.1103/PhysRevLett.110.148701

[12] E. Kao, V. Gadepally, M. Hurley, M. Jones, J. Kepner, S. Mohindra, P. Monticciolo, A. Reuther, S. Samsi, W. Song, D. Staheli, and S. Smith, "Streaming graph challenge: Stochastic block partition," in *2017 IEEE High Performance Extreme Computing Conference (HPEC)*. Waltham, MA: IEEE, 9 2017, pp. 1–12. [Online]. Available: doi.org/10.1109/HPEC.2017.8091040

[13] T. P. Peixoto, *Descriptive vs. inferential community detection in networks: pitfalls, myths, and half-truths*. Cambridge: Cambridge University Press, 11 2023. [Online]. Available: doi.org/10.1017/9781009118897

[14] T. P. Peixoto and A. Kirkley, "Implicit models, latent compression, intrinsic biases, and cheap lunches in community detection," *Physical Review E*, vol. 108, no. 2, p. 024309, 8 2023. [Online]. Available: doi.org/10.1103/PhysRevE.108.024309

[15] F. Wanye, V. Gleyzer, and W.-c. Feng, "Fast Stochastic Block Partitioning via Sampling," in *2019 IEEE High Performance Extreme Computing Conference (HPEC)*. Waltham, MA, USA: IEEE, 9 2019, pp. 1–7. [Online]. Available: doi.org/10.1109/HPEC.2019.8916542

[16] L. Durbeck and P. Athanas, "DPGS Graph Summarization Preserves Community Structure," *2021 IEEE High Performance Extreme Computing Conference, HPEC 2021*, 2021. [Online]. Available: doi.org/10.1109/HPEC49654.2021.9622846

[17] F. Wanye, V. Gleyzer, E. Kao, and W.-c. Feng, "SamBaS: Sampling-Based Stochastic Block Partitioning," *arXiv*, 8 2021. [Online]. Available: doi.org/10.48550/arXiv.2108.06651

[18] ——, "On the Parallelization of MCMC for Community Detection," in *Proceedings of the 51st International Conference on Parallel Processing*. New York, NY, USA: ACM, 2022, pp. 1–13. [Online]. Available: doi.org/10.1145/3545008.3545058

[19] A. J. Uppal, G. Swope, and H. H. Huang, "Scalable stochastic block partition," in *2017 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 9 2017, pp. 1–5. [Online]. Available: doi.org/10.1109/HPEC.2017.8091050

[20] F. Wanye, V. Gleyzer, E. Kao, and W.-c. Feng, "Exact Distributed Stochastic Block Partitioning," 5 2023. [Online]. Available: doi.org/10.48550/arXiv.2305.18663

[21] B. Karrer and M. E. J. Newman, "Stochastic blockmodels and community structure in networks," *Physical Review E*, vol. 83, no. 1, p. 016107, 1 2011. [Online]. Available: doi.org/10.1103/PhysRevE.83.016107

[22] T. P. Peixoto, "Entropy of stochastic blockmodel ensembles," *Physical Review E*, vol. 85, no. 5, p. 056122, 5 2012. [Online]. Available: doi.org/10.1103/PhysRevE.85.056122

[23] M. Besta, S. Weber, L. Gianinazzi, R. Gerstenberger, A. Ivanov, Y. Oltchik, and T. Hoefler, "Slim graph," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. New York, NY, USA: ACM, 11 2019, pp. 1–25. [Online]. Available: doi.org/10.1145/3295500.3356182

[24] J. Leskovec and C. Faloutsos, "Sampling from large graphs," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '06*. New York, New York, USA: ACM Press, 2006, p. 631. [Online]. Available: doi.org/10.1145/1150402.1150479

[25] T. Wang, Y. Chen, Z. Zhang, T. Xu, L. Jin, P. Hui, B. Deng, and X. Li, "Understanding Graph Sampling Algorithms for Social Network Analysis," in *2011 31st International Conference on Distributed Computing Systems Workshops*. IEEE, 2011, pp. 123–128. [Online]. Available: doi.org/10.1109/ICDCSW.2011.34

[26] A. S. Maiya and T. Y. Berger-Wolf, "Sampling community structure," in *Proceedings of the 19th international conference on World wide web - WWW '10*. New York, New York, USA: ACM Press, 2010, p. 701. [Online]. Available: doi.org/10.1145/1772690.1772762

[27] A. Terenin, D. Simpson, and D. Draper, "Asynchronous Gibbs Sampling," in *International Conference on Artificial Intelligence and Statistics*. Palermo: PMLR, 6 2020, pp. 144–154. [Online]. Available: doi.org/10.48550/arXiv.1509.08999

[28] A. J. Uppal, J. Choi, T. B. Rolinger, and H. Howie Huang, "Faster Stochastic Block Partition Using Aggressive Initial Merging, Compressed Representation, and Parallelism Control," *2021 IEEE High Performance Extreme Computing Conference, HPEC 2021*, 2021. [Online]. Available: doi.org/10.1109/HPEC49654.2021.9622836

[29] E. K. Kao, S. T. Smith, and E. M. Airoldi, "Hybrid Mixed-Membership Blockmodel for Inference on Realistic Network Interactions," *IEEE Transactions on Network Science and Engineering*, vol. 6, no. 3, pp. 336–350, 7 2019. [Online]. Available: doi.org/10.1109/TNSE.2018.2823324

[30] A. Strehl and J. Ghosh, "Cluster ensembles — a knowledge reuse framework for combining multiple partitions," *The Journal of Machine Learning Research*, vol. 3, no. 3, pp. 583–617, 3 2003. [Online]. Available: dl.acm.org/doi/10.1162/153244303321897735