— Dynamic Massive Parallel Computation Model for Graph Problems —

Isabelle Liu
School of Computer Science
Carleton University
Ottawa, Canada K1S 5B6
isabelleliu@cmail.carleton.ca

December 5, 2019

Abstract

The Massive Parallel Computation (MPC) model is becoming increasingly popular during the past few years due to its ability to satisfy the needs of modern applications that often require performing computations on data of massive scale. However, since the datasets in real world are constantly evolving, the static datasets assumption of MPC seems to be a critical limitation. In order to overcome this problem, studies of dynamic algorithms in the MPC model are conducted, and the Dynamic Massive Parallel Computation (DMPC) model is introduced. The DMPC model strengthens the MPC model by maintaining a solution to a problem more efficiently than recomputing the solution from scratch with the static algorithm. And this extension allows us to achieve, for algorithms of new graph problems, much lower computational complexity compared to that in the classic MPC model. In particular, we show that a maximal independent set can be maintained with $O(1)$ rounds, $O(1)$ active machines, and $O(\sqrt{N})$ communication per round.

1 Introduction

Big data and data analytics have gained popularity over the last decade, with the growing volume and variety of available data, and cheaper and more powerful computational processing. However, traditional data mining algorithms are becoming insufficient when performing computation on massive amounts of data, so parallel computing is introduced to increase the available computation power and to speed up application processing [1]. Parallel computing is a type of computing architecture where applications, computations, or processes are executed simultaneously. It basically breaks a larger task into several very similar smaller tasks so that they can be run independently at the same time, and their results are later combined upon completion. Parallel computing is widely used in various fields, such as climate prediction, fluid dynamics, and cryptology, in order to resolve their needs for speed.

One of the most popular models of parallel computing is the Massive Parallel Computation (MPC) model, which is seen as the canonical model for dealing with large scale data nowadays. It consists of machines with a large number of processors performing a set of coordinated tasks in synchronism [10]. And the main objective in the analysis of algorithms in the MPC model is to solve a problem with constant number of rounds while minimizing the amount of communication performed per round [9].

Although the MPC model is able to process massive amounts of data, it takes up large volumes of resources during computation. Also, because the datasets in the real world evolve continuously instead of being static, algorithms and models are forced to be re-executed after small modifications occur to the data, which leads to excessive processing time and resource requirements. As a result, researches of dynamic algorithms in traditional computing models are greatly promoted, because they are able to adjust and maintain solutions to given problems throughout input data modifications by performing only limited computation.

1.1 Problem Statement

In this project, I aim to implement and test the dynamic algorithm in the Dynamic Massive Parallel Computation (DMPC) model by Italiano et al. [13] on an alternative graph-theoretic problem, the maximal independent set (MIS). An MIS of a graph is one of the most well-studied problems in distributed and parallel settings and has diverse applications. Therefore, the ability to obtain and maintain an MIS efficiently is extremely crucial. My experiment validates the usefulness and importance of the dynamic algorithm since it shows a significant speed up from the classic MPC algorithm.

1.2 Organization of the Paper

In section 2, we will review the relevant literature and background. Then, section 3 will describe our model, the terminology that will be used in the paper, and the preprocessing process. In section 4, we will present the maximal independent set algorithm. Finally, section 5 will conclude the paper.

2 Literature Review

2.1 Classic MPC

The classic MPC model was first introduced in [14] as the MapReduce Class ($\mathcal{MRC}$), a computational model using MapReduce paradigm that interleaves sequential and parallel computation. In order for it to be practical, the model limited the number of machines and the memory per machine to be substantially sublinear in the size of the input. However, they placed very loose restrictions on the computational power of any individual machine, allowing it to run in polynomial time.

The model was later refined. For a set of μ machines $M_1, ..., M_\mu$ that exchange messages in synchronous rounds, each machine is able to send and receive messages of up to the size of its local memory, which is S bits, at each round. We assume that $S, \mu \in O(N^{1-\epsilon})$, where N is the size of the input and ϵ is efficiently small, then the overall amount of memory available across all machines in the system is bounded by $S \cdot \mu \in O(N^{2-2\epsilon})$, thus the same restriction applies to the total communication per round. For any MPC algorithms, there are three parameters that need to be bounded:

- Machine Memory: The total memory used by each machine per round is $O(N^{1-\epsilon})$.

- Total Memory: The total amount of data communicated per round is $O(N^{2-2\epsilon})$.
- Rounds: The number of rounds is $O(\log^i n)$, for a small $i \geq 0$.

2.2 MPC for Graph Problems

The concept of MPC is extensively studied in recents year. However, there exists a very important bottleneck in designing efficient MPC algorithms for graph problems when the space per machine is much smaller than the number of vertices n - it requires $\Omega(\log k)$ rounds to find a vertex at distance k from a given vertex [6]. For example, based on the 2-CYCLE conjecture, even distinguishing a graph consisting of single cycle of length n from 2 cycles of size $n/2$ requires $\Omega(\log n)$ rounds [15] [16]. Another one of the graph problem in the MPC model is the minimum spanning tree, which was recently proved to be solved in $O(\log n)$ rounds using an overall space of $\tilde{O}(n^{1+\epsilon})$ for some constant $\epsilon < 1$ [5].

It took almost 10 years for researchers to overcome the $O(\log n)$ barrier for computing approximate matching problems, and only very recently did Ghaffari and Uitto [12] implement an algorithm that can compute a $(1+\epsilon)$-approximate matching in $\tilde{O}(\sqrt{\log \Delta})$ rounds using sub-linear memory, where Δ is the maximum degree in the graph. Under the same memory assumption, Andoni et al. [2] showed that connected components was able to be run in $\tilde{O}(\log D)$ rounds, where D is the diameter of the graph. In addition, the computation of maximal independent set can be solved by the algorithm in [12] also in $\tilde{O}(\sqrt{\log \Delta})$ rounds. And this result is now improved by themselves to simple $O(\log \log \Delta)$ round with $\tilde{O}(n)$ memory per machine [11].

2.3 Dynamic Algorithm for Maximal Independent Set

A dynamic graph algorithm is called *incremental* if it allows edge insertions only, *decremental* if it allows edge deletions only, and *fully-dynamic* if it allows an intermixed sequence of both edge insertions and edge deletions [13]. A simple greedy algorithm can compute an MIS of a static graph in $O(m)$ time, where m is the number of edges in the graph. As such, an MIS can be trivially maintained in $O(m)$ time by simply recomputing it from scratch after each update.

The first dynamic MIS algorithm was given by Censor-Hillel et al. [8], which required $\Omega(\Delta)$ update time in the sequential setting. Later in a breakthrough, Assadi et al. [3] presented a deterministic fully-dynamic algorithm requiring $O(min\{\Delta, m^{3/4}\})$ amortized time, breaking the natural $\Omega(m)$ barrier for all graphs. This result was further improved in a series of subsequent researches, leading to a randomized algorithm also by Assadi et al. [4] that needed $\tilde{O}(min\{\sqrt{n}, m^{1/3}\})$ update time. Currently, the best algorithm maintains an MIS of a fully-dynamic graph in polylogarithmic time [7]. It takes $O(\log^2 \Delta \log^2 n)$ expected time per update, and can be adjusted to have $O(\log^2 \Delta \log^4 n)$ worst-case update-time with high probability.

2.4 Dynamic Algorithm for MPC

Dynamic algorithms maintain a solution to a given problem throughout a sequence of modifications to the input data. For a dynamic algorithm, the objective is to minimize the time spent and sometimes even the space required for updating the solution to a problem while the input gets modified. Unlike the classic MPC model, which has strictly sublinear memory of $\Omega(n)$ and requires $O(\log n)$ rounds to recompute a solution, the dynamic algorithm in the DMPC model hopes to establish some bounds for certain characteristics during recomputation:

- The number of machines active in each round
- The total amount of communication in each round
- The number of rounds required to update the solution

We can see that by bounding the number of active machines involved in the communication, we also imply the same bound on the amount of data that are sent in one round. While an ideal dynamic algorithm, which processes an input update in a constant number of rounds, using constant number of machines and constant amount of total communication, is often hard to achieve, a dynamic algorithm should at least use polynomially less resources than a static MPC model [13].

3 The Model

3.1 Terminology

Let $G = (V, E)$ be a graph, where V is the set of vertices and E is the set of edges in the graph. The number of vertices in the graph is $n = |V|$, and the number of edges is $m = |E|$. In this work, we reference the *lexicographically first maximal independent set* (LFMIS) that was used by Behnezhad [7]. It is obtained according to a ranking $\pi : V \to [0, 1]$ over the vertices in V. Initially, every vertex in V is alive. We iteratively take the alive vertex v with the minimum rank $\pi(v)$, add v to the MIS, and kill v and all of its alive neighbors. Therefore, v joins the MIS if none of its already processed neighbors have joined it before. We define several terms to describe each vertex:

- $LFMIS(G, \pi)$: The subset of all vertices that joined the MIS
- $deg(v)$: The degree of vertex v
- $count(v)$: The number of neighbors of vertex v that are in the MIS
- $N^+(v)$: The set of neighbors of v in the MIS
- $N^-(v)$: The set of neighbors of v not in the MIS
- ID: Assign an ID $(1, ..., n)$ to each vertex based on its ranking π

3.2 MPC Model

Since the dynamic algorithm in our model takes a graph as an input, the input size N is equal to $n + m$, which is the total number of vertices and edges in the graph. In addition, our algorithm is required to use a very limited amount of memory in each machine. Specifically, as our input is of size N, each machine is allowed to use only $O(\sqrt{N})$ memory. As a result, we make use of $O(\sqrt{N})$ machines. The computation proceeds in rounds. In each round, the $O(\sqrt{N})$ machines receive messages from the previous round and process the data stored in their own memory without communicating with each other. Then, each machine sends messages to other machines. And at the end of the computation, the output is stored across the different machines and is output collectively. The data output by each machine is at most the size of $O(\sqrt{N})$ because it has to fit in the local memory of each machine.

We have one of the machines act as a coordinator, denoted by M_C, and it stores an update-history $\mathcal{H}$ of the last $O(\sqrt{N})$ updates in both the input and the maintained solution, i.e., which edges have been inserted into and deleted from the input in the last $\sqrt{N}$ updates and which vertices have been inserted into and deleted from the maintained $LFMIS(G, \pi)$. All updates are sent to this single, arbitrary, but fixed machine that keeps additional information on the status of the maintained solution, and it then coordinates the rest of the machines to perform the update, by sending them large messages containing the additional information that it stores.

We also dedicate $O(n/\sqrt{N})$ machines to store statistics about the vertices of the graphs, such as their degree, whether they are in the MIS, how many of their neighbors are in the MIS, which of their neighbors are in or not in the MIS, and the machine storing their neighbors. In order to keep track of which machine keeps information about which vertices, many vertices with consecutive IDs are allocated together to a single machine so that we can store the range of IDs that are stored in each machine. Hence in M_C, except for the update history $\mathcal{H}$, we also store for each range of vertex IDs the machine that contains their statistics. Finally, M_C also stores the memory available in each machine.

4 Fully-Dynamic DMPC Algorithm for Maximal Independent Set

In this section, we apply the deterministic fully-dynamic DMPC algorithm to the maximal independent set. Our algorithm demonstrates that when restricting the memory of each machine to $\Omega(\sqrt{N})$ bits, the maximal independent set can be maintained in a constant number of rounds per update, a constant number of active machines per round, and a total communication of $O(\sqrt{N})$ per round.

For each update to the graph, we assume that the update history $\mathcal{H}$ is updated automatically. In addition, the update on the statistics of a vertex, such as its degree, whether it is in the MIS, the machine storing its alive neighboring vertices, etc., can be done in $O(1)$ rounds using a message through the coordinator machine M_C. Then after each update, we update the information stored in a machine by executing those updates in a round-robin fashion. And since we use $O(\sqrt{N})$ machines in the algorithm, each machine can be updated after at most $O(\sqrt{N})$ updates.

4.1 Supporting Function

We propose a set of supporting procedures to help with the edge updates and to guide the vertex allocation into machines throughout the sequence of updates.

- $getNbr(x)$: Returns the ID of the machine that stores the neighboring vertices of x.
- $getDegInMachine(M, x)$: Returns the degree of x in machine M.

- $fits(M, s)$: Return true if s vertices fit into machine M, and false otherwise.
- $toFit(s)$: Returns the ID of a machine that has enough memory to store s vertices, and the available space in that machine.
- $addEdge((x, y))$: We only describe the procedure for x, as the case for y is completely analogous. If y fit into $getNbr(x)$, we simply add y into $getNbr(x)$. If y does not fit in $getNbr(x)$, then call $moveVertices(x, s, M_x, toFit(s), \mathcal{H})$, where s is the number of vertices of x. If all of the remaining vertices in the machine M_x (of vertices other than x) fit into another machine, then move them there, in order to bound the number of used machines.
- $moveVertices(x, s, M_1, M_2, \mathcal{H})$: First, remove from machine M_1 deleted vertices of x based on $\mathcal{H}$. Second, send from M_1 up to s vertices of x to M_2. If the s vertices do not fit into M_2, move the neighbors of x from M_2 to a machine that fits them, i.e., execute $M_{x'} = toFit(s+getDegInMachine(M_2, x))$, and call $moveVertices(x, s, M_1, M_{x'}, \mathcal{H})$ and $moveVertices(x, getDegInMachine(M_2, x), M_2, M_{x'}, \mathcal{H})$.
- $updateVertex(x, \mathcal{H})$: Update the neighbors of x that are stored in $M_x = getNbr(x)$ based on $\mathcal{H}$. If the set of vertices of x does not fit in M_x after the update, call $moveVertices(x, s, M_x, toFit(s), \mathcal{H})$, where s is the number of vertices of x. If all of the remaining vertices in the machine M_x (of vertices other than x) fit into another machine, then move them there, in order to bound the number of used machines.
- $updateMachine(M, \mathcal{H})$: Update all adjacency lists stored in machine M to reflect the changes in the update history $\mathcal{H}$. If all of the remaining vertices of the machine fit into another half-full machine, then move them there, in order to bound the number of used machines.

4.2 Handling Update

Now we explain how our algorithm updates the maintained maximal independent set after an edge update.

4.2.1 Insertion

On insertion of an edge (x, y), update is required only in case both vertices are in the MIS.

1. Execute $updateVertex(x, \mathcal{H})$, $updateVertex(y, \mathcal{H})$, and $addEdge((x, y))$.
2. If $\pi(x) < \pi(y)$, keep x in $LFMIS(G, \pi)$ and remove y from $LFMIS(G, \pi)$.
3. If there is a neighbor u of y with $\pi(u) > \pi(y)$ and none of whose neighbors are in the MIS, add all such u to $LFMIS(G, \pi)$. Otherwise, do nothing and return.
4. If $\pi(y) < \pi(x)$, proceed analogously.
5. In any case, the update-history is updated to reflect all the changes caused by the insertion of (x, y).

4.2.2 Deletion

In case of deletion of an edge (x, y), update is required when only one of them is in the MIS.

1. Call $updateVertex(x, \mathcal{H})$ and $updateVertex(y, \mathcal{H})$.
2. Assume $z \in \{x, y\}$ is not in $LFMIS(G, \pi)$.
3. If no neighbors u of z with $\pi(u) < \pi(z)$ are in the MIS, add z into $LFMIS(G, \pi)$. Otherwise, do nothing and return.
4. If a neighbor v of z with $\pi(v) > \pi(z)$ is in $LFMIS(G, \pi)$, remove it from the MIS.
5. Check if there exists a neighbor w of v whose neighbors are all not in the MIS; if yes, add all such w to $LFMIS(G, \pi)$.
6. In any case, the update-history is updated to reflect all the changes caused by the deletion of (x, y).

4.3 Algorithmic Complexity

In our dynamic algorithm, both the insertion and deletion of an edge run in $O(1)$ rounds, use $O(1)$ machines, and generate $O(\sqrt{N})$ communication per round. For each update, we can access the machine that stores the neighboring vertices of a vertex x in $O(1)$ rounds, and only a fixed number of vertices need to be updated. All supporting functions are trivially executable in $O(1)$ rounds. In addition, as each machine is updated every $O(\sqrt{N})$ rounds, it follows that the number of edges that have been inserted into or removed from the graph and the machines storing those edges that are not yet updated, is $O(\sqrt{N})$. And since all the calls to $moveVertices$ transfer at most $O(\sqrt{N})$ vertices of x due to the memory limitation for each machine being $O(\sqrt{N})$, there is at most a constant number of calls to $moveVertices$.

5 Conclusions

We have implemented the dynamic algorithm in the Dynamic Massive Parallel Computation model by Italiano et al. [13], an extension of the widely popular Massive Parallel Computation algorithm, and tested it on a fundamental graph problem, maximal independent set. We present that this fully-dynamic algorithm maintains a maximal independent set in $O(1)$ rounds per update, while the number of machines that are active per round is $O(1)$, and the total communication per round is $O(\sqrt{N})$. As future research, it would be interesting to apply the algorithm to other problems, such as the graph coloring.

References

[1] Parallel computing — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Parallel_computing`, 2019. [Online; accessed 26-September-2019].

[2] A. Andoni, Z. Song, C. Stein, Z. Wang, and P. Zhong. Parallel graph connectivity in log diameter rounds. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 674–685, 2018.

[3] Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. Fully dynamic maximal independent set with sublinear update time. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2018, pages 815–826, New York, NY, USA, 2018. ACM.

[4] Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. Fully dynamic maximal independent set with sublinear in n update time. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '19, pages 1919–1936, Philadelphia, PA, USA, 2019. Society for Industrial and Applied Mathematics.

[5] M. H. Bateni, S. Behnezhad, M. Derakhshan, M. T. Hajiaghayi, and V. Mirrokni. Massively parallel dynamic programming on trees, 2018.

[6] S. Behnezhad, L. Dhulipala, H. Esfandiari, J. Łącki, V. Mirrokni, and W. Schudy. Massively parallel computation via remote memory access. In *The 31st ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '19, pages 59–68, New York, NY, USA, 2019. ACM.

[7] Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Cliff Stein, and Madhu Sudan. Fully dynamic maximal independent set with polylogarithmic update time, 2019.

[8] Keren Censor-Hillel, Elad Haramaty, and Zohar Karnin. Optimal dynamic distributed MIS. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, PODC '16, pages 217–226, New York, NY, USA, 2016. ACM.

[9] D. Durfee, L. Dhulipala, J. Kulkarni, R. Peng, S. Sawlani, and X. Sun. Parallel batch-dynamic graphs: Algorithms and lower bounds, 2019.

[10] S. E. Fahlman, G. E. Hinton, and T. J. Sejnowski. Massively parallel architectures for AI: NETL, thistle, and boltzmann machines. pages 109–113, 1983.

[11] M. Ghaffari, T. Gouleakis, C. Konrad, S. Mitrović, and R. Rubinfeld. Improved massively parallel computation algorithms for MIS, matching, and vertex cover, 2018.

[12] M. Ghaffari and J. Uitto. Sparsifying distributed algorithms with ramifications in massively parallel computation and centralized local computation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '19, pages 1636–1653, Philadelphia, PA, USA, 2019. Society for Industrial and Applied Mathematics.

[13] G. F. Italiano, S. Lattanzi, V. S. Mirrokni, and N. Parotsidis. Dynamic algorithms for the massively parallel computation model. In *The 31st ACM Symposium on Parallelism*

in Algorithms and Architectures, SPAA '19, pages 49–58, New York, NY, USA, 2019. ACM.

[14] Howard Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for mapreduce. In *Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, pages 938–948, Philadelphia, PA, USA, 2010. Society for Industrial and Applied Mathematics.

[15] T. Roughgarden, S. Vassilvitskii, and J. R. Wang. Shuffles and circuits: (on lower bounds for modern parallel computation). In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '16, pages 1–12, New York, NY, USA, 2016. ACM.

[16] G. Yaroslavtsev and A. Vadapalli. Massively parallel algorithms and hardness for single-linkage clustering under ℓ_p-distances. In *35th International Conference on Machine Learning*, ICML '18, 2018.