Parallel Algorithms: The MPC Model

1 Introduction

Throughout this book we have dealt with *sequential* computations. In this chapter we look at *parallel* computations.

There are many models of parallelism. One of the first ones was the PRAM (Parallel Random Access Machine) which allowed the processors to have access to a shared memory. This feature (or bug) leads to the need for subcategories of PRAM depending on if concurrent reads or concurrent writes are allowed. There is a vast literature on both algorithms and lower bounds for PRAMs. For algorithms we recommend Karp's survey [**?**] and the papers it references. For lower bounds we recommend the papers of Cook & Dwork [**?**] and Li & Yesha [**?**] and the papers they reference.

We study a more recent model called *Massively Parallel Computation*. We will define it, give examples of problems it solves well, and then discuss lower bounds on it. We will note some relations to the PRAM when they come up. When we refer to PRAMs we will mean those that allow concurrent reads and writes. We omit details about how they resolve contentions.

2 The Massively Parallel Computing Model (MPC)

Beame et al. [**?**] invented the following model.

Definition 1. *The* Massively Parallel Computation model (MPC) *consists of the following:*

- *The input data size N. (For a graph $G = (V, E)$ this is $\max\{|V|, |E|\}$ which is usually $|E|$.)*
- *The number of machines M available for computation. The machines will communicate with each other in rounds (we elaborate on this later).*
- *The memory size, s words, each machine can hold.*

A reasonable assumption here is that a single word stores a constant number of bits. Thus in practice, each machine is capable of storing $\Theta(s)$ number of bits. Moreover, in the literature, it is most often assumed that

$$M \cdot s = \Theta(N). \tag{1}$$

This is the most interesting scenario since the number of available resources for the algorithm ($M \cdot s$) is asymptotically equal to the size of the input data.

Input and output work as follows.

1. The input data is split across the M machines *arbitrarily*. If the input is a graph then initially each edge is given to some machine. Note that a machine may well have many edges.
2. There will be some machines designated as *output machines*. At the end of the computation the answer will be stored there in a distributive manner. We give an example. If the problem is connected components then we want to assign to every vertex v a number n_v such that two vertices are in the same component iff they are assigned the same number. In addition to the input machines there will be n output machines, one for each vertex v, and at the end machine v has n_v.

Computation is performed in synchronous rounds. In each round, every machine performs some computation on the data that resides locally, then sends/receives messages to any other machine. Since the local memory of each machine is bounded by s words, we assume that a single machine can send and receive at most s words per round; however, each of these words can be addressed to or received from different machines. The two main parameters that are investigated in this model are:

- The number of communication rounds it takes for an algorithm to solve a problem, often called *running time*. The local computations are ignored in the analysis of the running time MPC algorithms because communication is the bottleneck. In practice, these local computations frequently run in linear or near-linear time, however, there is no bound on their length formally.
- The size of local memory s. Problems are easier to solve with larger s. In the extreme when $s \gg N$, one can just put the entire input on a single machine and solve it locally. Typically, s is polynomially smaller than N, e.g. $s = N^\epsilon$ for some constant $\epsilon < 1$.

If one of the machines had most of the input then, since we allow machines unlimited power, the problem could likely be done rather quickly. This is not what we want to model. We want to model problems where the input is so large that no machine can have most of it. Hence we have the following definition.

Definition 2. *An* MPC *algorithm is* strongly sublinear *if each machine gets space $s \ll N$. So the space is much less than the input size. For graph problems (on dense graphs) it will mean that there exists $\delta < 1$ such that $s \leq n^{1+\delta}$. We will often use the term* strongly sublinear *rather than specify the space precisely.*

Note the following

Fact 1.

1. *The* MPC *algorithm is non-uniform. For every N we have a different* MPC *algorithm for input length N. These algorithms will be very similar.*
2. *An* MPC*-computation can be viewed as a directed graph in layers. The first layer has the input machines. The second layer has the machines that the first layer communicates with. Put a directed edge between a machine in the first layer and a machine that it sends to. Keep doing this for layers $2, 3, \ldots, R+1$. In Section* **??** *we use this viewpoint.*
3. *Any PRAM algorithm working in time t can be simulated by an* MPC *algorithm in $O(t)$ rounds and with strongly sublinear memory per machine.*

3 Some Algorithms in the MPC Model

3.1 Connected Components

Problem 3.1. *Connected Components (*CC*)*
INSTANCE: An undirected graph $G = (V, E)$.
QUESTION: Which vertices are in the same connected component? A solution is a labeling of vertices $\ell(v)$ such that $\ell(u) = \ell(v)$ if and only if vertices u and v are in the same connected component.

The runtime of an algorithm for CC will depend on the number of vertices n, the number of edges m, and a new parameter, the diameter of the graph D, which we define.

Definition 3. *Let G be a graph. The* diameter D of G *is the length of the longest shortest path between two vertices. Formally:*

$$D = \max_{u,v \in V} \textit{The length of the shortest path from } u \textit{ to } v.$$

Behnezhad et al. [**?**] proved showed the following theorem.

Theorem 1. *For all ϵ the following holds. There is a randomized strongly sublinear* MPC *algorithm for* CC *such that, on a graph $G = (V, E)$ ($|V| = n$, $|E| = m$, Diameter D):*

1. *The total space used is $O(m)$*
2. *The number of rounds is $O(\log D + \log \log_{m/n}(n))$.*
3. *The algorithm succeeds with high probability.*
4. *The algorithm does not need to know D.*

We will not provide a proof of Theorem **??**. Instead, we will give a short overview of a slightly weaker result due to Andoni et al. [**?**].

Theorem 2. *There is a randomized strongly sublinear* MPC *algorithm for* CC *such that, on a graph $G = (V, E)$ ($|V| = n$, $|E| = m$, Diameter D):*

1. *The total space used is $O(m)$*
2. *The algorithm takes $O(\log D \cdot \log \log_{m/n}(n))$ rounds.*
3. *The algorithm succeeds with high probability.*
4. *The algorithm does not need to know D.*

Behnezadi et al. [**?**] write the following about the ideas which lead to Theorem **??**:

Graph exponentiation.

> Consider a simple algorithm that connects every vertex to vertices within its 2-hop (i.e., vertices of distance 2) by adding edges. It is not hard to see that the distance between any two vertices shrinks by a factor of 2. By repeating this procedure, each connected component becomes a clique within $O(\log D)$ steps. The problem with this approach, however, is that the required memory of a single machine can be up to $\Omega(n^2)$, which for sparse graphs is much larger than $O(m)$.

Solution to CC **Built off the Graph Exponentiation Technique.**

> Suppose that every vertex in the graph has degree at least $d \gg \log n$. Select each vertex as a leader independently with probability $\Theta(\frac{\log n}{d})$. Then contract every non-leader vertex to a leader in its 1-hop (which w.h.p. exists). This shrinks the number of vertices from n to $O(n/d)$. As a result, the amount of space available per remaining

vertex increases to $\Omega(\frac{m}{n/d}) = \Omega(\frac{nd}{n/d}) = d^2$. At this point, a variant of the aforementioned graph exponentiation technique can be used to increase vertex degrees to d^2 (but not more), which implies that another application of leader contraction decreases the number of vertices by a factor of $\Omega(d^2)$. Since the available space per remaining vertex increases doubly exponentially, $O(\log \log n)$ phases of leader contraction suffice to increase it to n per remaining vertex. Moreover, each phase requires $O(\log D)$ iterations of graph exponentiation, thus the overall round complexity is $O(\log D \cdot \log \log n)$.

3.2 Maximal Independent Set.

Problem 3.2. *Maximal Independent Set (*MIS*)*
INSTANCE: A graph $G = (V, E)$.
QUESTION: Return an independent set I such that no independent set is a proper superset of I.

Note the following:

- The *Maximal Ind. Set Problem* is very different from the *Maximum Ind. Set Problem*. Maximal Ind. Set is in P by a simple greedy algorithm. Maximum Ind. Set is NP-complete.
- The greedy algorithm for MIS is inherently sequential. Hence it is not obvious that there is a fast parallel algorithm for MIS. However, there is.

Luby [**?**] gave a framework for MIS algorithms for the PRAM.
Luby's algorithm for MIS**:**

1. Fix a random permutation $\pi : [n] \to [n]$ of vertices.
2. A vertex v adds itself to I if, for all neighbors u of v, $\pi(v) < \pi(u)$.
3. Remove selected vertices and their neighbors.
4. Repeat until reaching an empty graph.

The following theorem comes from a simple analysis of the above method. The details can be found in Luby's paper.

Theorem 3. *There exists an algorithm in the PRAM model which, given a graph G on n vertices and m edges, solves* MIS *in $O(\log n)$ depth and $O(m)$ work.*

Ghaffari & Uitto [**?**] showed that the local nature of the MIS problem can be efficiently exploited in the MPC model. Namely they proved the following.

Theorem 4. *Let $\epsilon \in (0, 1)$. There is a randomized* MPC *algorithm for* MIS *with the following properties:*

1. *Each machine uses $O(n^\epsilon)$ memory.*
2. *The number of rounds is $O(\sqrt{\log n} \cdot \log \log n)$ (the constant on the O depends on ϵ).*
3. *The probability that an* MIS *is found is $\geq 1 - \frac{1}{10n}$*

Theorem **??** is the best-known result for general graphs. Ghaffari, Grunau, Jin [**?**] showed that the MIS problem, restricted to some specific class of graphs (i.e. trees or graphs with bounded arboricity), has a randomized strongly sublinear MPC algorithm working in $O(\log\log n)$ rounds. Ghaffari, Kuhn, Uitto [**?**] showed a (conditional) lower bound on the MIS problem in the MPC model of $\Omega(\log\log n)$ rounds. This is the best known lower bound on MIS in the MPC model.

3.3 Fast Fourier Transform.

Hajiaghayi et al. [**?**] obtained a constant-round MPC-algorithm for Fast Fourier Transform and used that to get a constant-round MPC-algorithm for pattern matching (with wildcards). (All results in this section are from that paper.)

Problem 3.3. *Fast Fourier Transform and String Matching*

INSTANCE: Complex numbers $x_0, \ldots, x_{n-1}$. (These will have rational real and complex parts so they are finite length.)

QUESTION: Return the n complex numbers $X_0, \ldots, X_{n-1}$ where

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi kn/N} \qquad k = \{0, \ldots, N-1\}.$$

(We can either use an approximation or keep the roots of unity in symbolic form so that the length is manageable.)

Hajiaghayi et al. [**?**] showed the following:

Theorem 5. *Let $\epsilon > 0$. There is a deterministic* MPC *algorithm for FFT with local memory $O(n^\epsilon)$, total memory $O(n\text{polylog}(n))$, and $O(\frac{1}{\epsilon})$ rounds.*

Problem 3.4. *Pattern Matching and Variants*

INSTANCE: A text T and a pattern P. They are both over an alphabet Σ. The text is usually much longer than the pattern. We describe three types of patterns in the QUESTION part.

QUESTION:

- *$P \in \Sigma^*$. We want all occurrences of P in T.*
- *$P \in \{\Sigma \cup ?\}^*$. The pattern occurs if it matches all of the characters in Σ. We do not care what happens at the ? spot. For example, acb?a occurs in* **acbaa***bbbbb***acbba** *as shown.*
- *$P \in \{\Sigma \cup +\}^*$. The pattern occurs if it matches all of the characters in Σ. At the + spots a single character can appear many times. For example, acb?a occurs in*

  acbaaaaaaaaa *bbbbb* **acbbbbbbbbbbbbbbbbbbba**

  as shown.
- *$P \in \{\Sigma \cup *\}^*$. The pattern occurs if it matches all of the characters in Σ. At the $*$ spots any string can appear. For example, acb?a occurs in*

  acbabbcaabbc *bbbbb* **acabaaaccabbaaadaacadaa**

  as shown.

Theorem 6. *Let* $0 < \epsilon < 1$.

1. *There exists an* MPC *algorithm for string matching with pattern* $P \in \Sigma^*$ *with* $M = O(n^\epsilon)$, $s = O(n^{1-\epsilon})$, *and* $r = O(1)$.
2. *There exists an* MPC *algorithm for string matching with* $P \in \{\Sigma \cup ?\}^*$ *with* $M = O(n^\epsilon \text{polylog}(n))$, $s = O(n^{1-\epsilon}\text{polylog}(n))$, *and* $r = O(1)$.

It is not known if pattern matching with $P \in \{\Sigma \cup *\}^*$ has a polylog round MPC algorithm with sublinear M and s.

4 Unconditional MPC Lower Bounds

In this section, we are going to present an approach to find unconditional lower bound for problems on the MPC model. This approach was introduced by Roughgarden et al. [**?**]. (Everything in this section is from that paper.) We will do the following:

1. Define the s-Shuffle model.
2. State (but not prove) a relation between the MPC model and the s-Shuffle model.
3. Show that an s-Shuffle computation can be represented by polynomials.
4. Obtain lower bounds on the s-Shuffle model for some problems by looking at polynomials.
5. Use these lower bounds and the relation between the MPC model and the s-Shuffle model to get lower bounds on the MPC model.

The lower bounds are not tight; however, they are important because they are unconditional.

4.1 The s-Shuffle Model.

In the s-Shuffle model, we have multiple machines. Each machine has s inputs and is located in one of R levels. These machines may seem like gates in a circuit but they are not. In particular, they can compute arbitrary functions, not just AND, OR, and NOT.

Each input of a machine at level R comes from a machine at a lower level. The inputs are 0, 1, or $\perp$ (we think of $\perp$ as meaning silence—no input). For each machine, at most one of its inputs can be other than $\perp$. In the following definition we describe how each input bit of each machine is determined.

Definition 4.

1. *The* $\perp$*-sum of* $z_1, z_2, \cdots, z_m \in \{0, 1, \perp\}$ *is:*
   - 1 *if exactly one* z_i *is* 1 *and the rest are* $\perp$*;*
   - 0 *if exactly one* z_i *is* 0 *and the rest are* $\perp$*;*
   - $\perp$ *if every* z_i *is* $\perp$*;*
   - *undefined (or invalid) otherwise*
2. *The* $\perp$*-sum of* s m*-tuples is computed component-wise.*

For each port of every machine, $\perp$-sum is used on signals received on that port to determine the output of the corresponding port. Now that we understand how each input handles multiple signals, we describe the s-Shuffle more formally.

Definition 5. *An R* rounds s-Shuffle *is a model with a set of V machines such that each machine v has a round number $0 \leq r(v) \leq R+1$. Round 0 indicates an input machine which receives input signals $x_1, x_2, \cdots, x_n$. Round $R+1$ indicates an output machine which receives output signals $y_1, y_2, \cdots, y_k$.*

1. *For all pairs $u, v \in V$ so that $r(u) < r(v)$, we have a function $\alpha_{u,v} : \{0,1,\perp\}^s \to \{0,1,\perp\}^s$ which defines signals that machine u will send to all inputs of v according to the output of u, i.e. machine u will send $\alpha_{u,v}(g(u))$ where $g(u)$ is the output of machine u. Note that during a computation a machine will get many s-tuples.*

2. *We now define how the s-Shuffle computes. Recall that a machine v receives many signals (actually s-tuples). So what will machine v take to be its input? It will take the $\perp$-sum of all signals received component-wise.*

Definition 6. *The* width *of an s-Shuffle is the maximum number of machines in a round other than rounds 0 and $R+1$.*

We state the connection between the MPC model and the s-Shuffle model.

Theorem 7. *An* MPC *computation that runs on M machines with space s in R rounds can be simulated with an s-Shuffle instance with width M in R rounds.*

4.2 Polynomials

Definition 7.

1. *We will be using polynomials on many variables. We will only care about what they output when the variables are replaced by 0's and 1's. Hence the polynomials will not have any exponents. Moreover, when we say $p \in \mathsf{Z}[x_1, \ldots, x_m]$ we will implicitly mean that there are no exponents.*

2. *The* degree *of a polynomial is the number of variables in the largest monomial.*

3. *Let $f : \{0,1\}^m \to \{0,1\}$. Let $p \in \mathsf{Z}[x_1, \ldots, x_m]$. p* represents *f if, for all $\vec{b} \in \{0,1\}^m$, $f(\vec{b}) = p(\vec{b})$.*

4. *If f is a graph property on graphs with n vertices then the variables are $x_{i,j}$ (with $1 \leq i < j \leq n$) and represent edges.*

Theorem 8. *For every $f : \{0,1\}^m \to \{0,1\}$ there exists $p \in \mathsf{Z}[x_1, \ldots, x_m]$ of degree m such that f represents g.*

Proof. For every $\vec{b} \in \{0,1\}^m$ create a polynomial that is 1 iff the input is $\vec{b}$. For example, if $n = 4$ and the bit sequence is 0110 then we associate the polynomial

$$\text{POLY}(\vec{b}) = (1 - x_1)x_2x_3(1 - x_4).$$

The polynomial p is

$$p(x_1, \ldots, x_m) = \sum_{\vec{b}:\ f(\vec{b})=1} \text{POLY}(\vec{b}).$$

□

Note We will connect the smallest degree of a polynomial that represents f to the complexity of f.

EXER Let AND_n be the function $x_1 \wedge \cdots \wedge x_n$. Let OR_n be the function $x_1 \vee \cdots \vee x_n$.

1. Give a polynomial of degree n that represents AND_n. Same for OR_n.
2. Show that any polynomial that represents AND_n has degree $\geq n$. Same for OR_n.
3. Show that any s-Shuffle for AND_n needs at least $\lceil \log_s d \rceil$ rounds. Same for OR_n.

END EXER

Now, we are going to show that the output of an s-Shuffle computation with n input bits, and k output bits, running in R rounds, can be produced by k polynomials with degree of at most s^R such that each polynomial produces one of the s-Shuffle outputs.

Theorem 9. *Let $f : \{0,1\}^n \to \{0,1\}^k$.*

1. *If there is an r-round s-Shuffle for f, then there are k polynomials $\{p_i(x_1, \cdots, x_n)\}_{i=1}^k$ of degree at most s^r such that, for all $x \in \{0,1\}^n$, for $1 \leq i \leq k$, $p_i(x) = f(x)_i$.*
2. *If f cannot be represented by a polynomial with degree less than d, then any s-Shuffle that computes f has least $\lceil \log_s d \rceil$ rounds. (This follows from Part 1).*

Proof. First, for every machine v and value $\vec{z} \in \{0,1,\perp\}^s$, we introduce a polynomial $p_{v,\vec{z}}(x_1, \cdots, x_n)$ which is 1 if the output of machine v will be $\vec{z}$ when the input of our s-Shuffle instance is $\vec{x} = (x_1, \cdots, x_n)$, and otherwise, it is 0. Now, we are going to prove that the degree of $p_{v,\vec{z}}(\vec{x})$ is at most $s^{r(v)}$.

The base of our induction is input machines. For each input x_i, we have a specific machine with polynomial defined as follows:

- $p_{v,\mathbf{z}}(x_1, \cdots, x_n) = 1 - x_i$ for $\mathbf{z} = (0, \perp, \cdots, \perp)$
- $p_{v,\mathbf{z}}(x_1, \cdots, x_n) = x_i$ for $\mathbf{z} = (1, \perp, \cdots, \perp)$
- $p_{v,\mathbf{z}}(x_1, \cdots, x_n) = 0$ otherwise

These polynomials have degree at most $s^0 = 1$. For the induction step, we assume that for all machines in level $0, \cdots, r(v)-1$ like u, there are polynomials with degree at most $s^{r(u)}$ to compute the $p_{u,\mathbf{z}}(\mathbf{x})$ for all $\mathbf{z}$. Now, we want to find polynomials for $p_{v,\mathbf{z}}$. For each bit of $\mathbf{z}$ we have two cases:

1. $z_i \in 0, 1$: In this case, port i of machine v should get z_i from one of the previous machines and $\perp$ from other machines. Therefore, for each machine u with $r(u) < r(v)$ and for each $z' \in \{0,1,\perp\}^s$ such that $\alpha_{u,v}(z')$ has value z_i at entry i, we sum these $p_{u,z'}$. Given that just one of them should send a value other than $\perp$, this summation should not create any problem, i.e. for every input its value should be in $\{0,1\}$.

2. $z_i = \perp$: In the previous case we find polynomials when $z_i = 0$ or 1. So 1 minus these polynomials should give the answer for the case $z_i = \perp$.

Both polynomials in these two cases are made by summation of some polynomials with degree at most $s^{r(v)-1}$. Hence, the final degrees of these polynomials are at most $s^{r(v)-1}$. In the end, for each $\mathbf{z}$, we set $p_{v,\mathbf{z}}$ equal to product of all polynomials which we obtain for each entry of $\mathbf{z}$. This will give a polynomial with degree at most $s^{r(v)}$. □

4.3 Lower Bounds in the MPC Model for Monotone Graph Properties

We will use two known results to obtain a lower bound on s-Shuffles for monotone graph properties.

Definition 8.

1. *A* monotone graph property *is a property of graphs such that if more edges are added then the property still holds.* Examples*: connectivity, non-planarity.*
2. *A* Decision Tree for a graph property *is a decision tree for the property where the queries are "$(i, j) \in E$?".*
3. *In this section (and only this section) a polynomial is over $\mathsf{Z}[x_1, \ldots, x_m]$.*

Theorem 10. *Let P be a monotone graph property.*

1. *Rosenberg [?] proved that any decision tree for P requires $\Omega(n^2)$ depth. (The constant for the Ω was increased by Rivest & Vuillemin [?] and Kahn et al. [?].)*
2. *Buhrman and de Wolf [?] proved that the decision tree complexity of a polynomial in $\mathsf{Z}[x_1, \ldots, x_n]$ of degree d is at most $O(d^4)$.*
3. *The degree of a polynomial representing a monotone graph property is at least $\sqrt{n}$. (This follows from Parts 1 and 2.)*
4. *Any s-Shuffle computation for P has at least $\Omega(\log_s n)$ rounds. (This follows from Part 3 and Theorem ??.)*

With more effort, the following has also been proved.

Theorem 11. *For the problem of graph connectivity the following hold.*

1. *Every s-Shuffle needs at least $\lceil \log_s \binom{n}{2} \rceil$ rounds.*
2. *Every* MPC *algorithm with space s needs at least $\lceil \log_s \binom{n}{2} \rceil$ rounds. (This follows from Theorem ??.)*

5 Conditional MPC Lower Bounds

In this section, we are going to present conditional lower bound for problems using an MPC algorithms. The framework for these lower bounds was introduced by Ghaffari et al. [?]. (Everything in this section is from that paper unless otherwise noted.) We will do the following:

1. Define the 1vs2-Cycle problem and formally state the conjectured lower bound for it in the MPC model.
2. Assuming the conjectured lower bound for the 1vs2-Cycle, and additional assumptions, we will state lower bounds for other problems in the MPC model.

Problem 5.1. 1vs2-Cycle
INSTANCE: An undirected graph $G = (V, E)$ which we are promised is either one cycle or the union of two cycles.
QUESTION: Determine if the graph is one cycle or the union of two cycles.

The following conjecture is widely believed:

Conjecture 1. The 1vs2-Cycle Conjecture *Any* MPC *algorithm for the* 1vs2-Cycle *problem using machines with $s = n^\epsilon$ requires $\Omega(f(\epsilon)\log n)$ rounds for some f. (Note that since the input is one cycle or two cycles, $m = O(n)$ so strongly sublinear now means $s = n^\epsilon$.)*

The lower bound is conjectured to hold even for the simpler promise problem of distinguishing whether an input graph is a cycle of length n or two cycles of length $n/2$.

Theorem 12. *Assume the* 1vs2-Cycle *conjecture. Any strongly sublinear* MPC *for undirected connectivity requires $\Omega(\log n)$ rounds.*

Proof. It is clear that if we have one cycle, the graph is connected, otherwise, the graph is unconnected. Thus, if we find an algorithm for undirected connectivity in $o(\log n)$ rounds, then we will have an algorithm in $o(\log n)$ rounds for 1vs2-Cycle, which contradicts the 1vs2-Cycle conjecture. □

We can try to find conditional lower bounds for other problems by making a reduction from 1vs2-Cycle or other problems that have a known conditional lower bound. But, alas, we need to introduce a restriction on the type of algorithms we can get lower bounds on.

Intuitively, a *component-stable* MPC *algorithm* is one where the output machines from different connected components (viewing the MPC algorithm as a graph) are independent. This notion only makes sense if the output machines are associated to vertices or edges of the graph. We give three examples of problems where we also carefully define the output machines.

Problem 5.2. *Maximal Matching*
INSTANCE: An undirected graph $G = (V, E)$
QUESTION: Return an independent set I such that no independent set is a proper superset of I.
OUTPUT: For every $v \in V$ there is an output machine. At the end of the computation the machine will have either a 1 (for $v \in I$) or a 0 (for $v \notin I$).

Problem 5.3. *Sinkless Orientation*
INSTANCE: An undirected graph $G = (V, E)$
QUESTION: Return an orientation of the graph (a direction for every edge) so that the graph has no vertices of outdegree 0. (Such vertices are called sinks*.)*
OUTPUT: For every $\{u, v\} \in E$ there is an output machine. At the end of the computation the machine will have either (u, v) or (v, u) to indicate the orientation.

Problem 5.4. $(\Delta+1)$-*Graph Coloring*

INSTANCE: An undirected graph $G=(V,E)$ *and its max degree* Δ.

QUESTION: Return a proper $(\Delta+1)$*-coloring of* G *(such always exists). Colors are* $\{1,\ldots,\Delta+1\}$.

OUTPUT: For every $v \in V$ *there is an output machine. At the end of the computation the machine will have the color of* v.

Definition 9. *Let* A *be a graph problem where the output is data attached to each vertex (a similar definition is used if the output is data attached to each edge).* A component-stable MPC algorithm *for* A *has the following property: Let* $u,v \in V$. *Let* M_u *and* M_v *be the output machines associated to* u,v. *View the* MPC *algorithm as a directed graph. If* u,v *are in different components of* G *then* M_u *and* M_v *are in different components of the* MPC *algorithm.*

The only known MPC algorithms for Maximal Matching, Sinkless Orientation, Δ-Graph Coloring, and many other graph problems are component-stable. Hence it is of interest to get lower bounds on component-stable MPC algorithms for these and other problems.

Theorem 13. *Assume the* 1vs2-Cycle *conjecture. Assume that all* MPC*'s discussed in this theorem are strongly sublinear and use a component stable algorithm.*

1. *Maximal Matching requires* $\Omega(\log\log n)$ *rounds. (Same holds for a constant approximation for Maximal Matching. This lower bound holds even when restricted to trees.)*
2. *Sinkless Orientation requires* $\Omega(\log\log\log n)$ *rounds.*
3. *Let* c *be a constant.* c*-coloring a cycle requires* $\Omega(\log\log^* n)$ *rounds.*
4. *Any constant approximation for* VC *requires* $\Omega(\log\log n)$ *rounds.*
5. *(Using additional assumptions)* $(\Delta+1)$*-coloring a graph requires* $O(\sqrt{\log\log n})$ *rounds.*

6 Future Directions

Despite the recent interest in the MPC model, there are many fundamental open problems. For example, the MPC-complexity of 2SAT and directed S-T connectivity remain unknown. These problems are related in the sense that solving 2SAT is usually done by reducing it to directed S-T connectivity.