Lower bounds in MPC **and** AMPC **models**
Samira Goudarzi, Peyman Jabbarzade, Jan Olkowski

1 Introduction

Throughout this book we have dealt with *sequential* computation. In this chapter we look at *parallel* computation. 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 [12] and the papers it references. For lower bounds we recommend the papers of Cook & Dwork [7] and Li & Yesha [13]; and the papers they reference.

Definition 1.

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 relation to the PRAM when they come up.

2 The Massively Parallel Computing Model (MPC)

Beame et al. [2] invented the following model.

Definition 2. *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 each machine can hold, s words.*

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 whole 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 locally 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*. Note that 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 3. *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 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.*

Fact 1. *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 in the MPC Model

3.1 Connected Components

Problem 3.1. *Connected Components*
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 connectivity 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 4. *Let G be a graph. The* diameter D of G *is the longest shortest path between two vertices. Formally:*

$$D = \max u, v \in V \textit{ The shortest path from } u \textit{ to } v.$$

Behnezhad et al. [3] proved showed the following theorem.

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

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

We will not provide proof of the two above results. Instead, we will give a short overview of a slightly weaker result due to Andoni et al. [1].

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

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. [3] write the following about the ideas which lead to Theorem 2:

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 Connectivity 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.

We note to points about MIS.

- Note that *Maximal Ind. Set* is very different from *Maximum Ind. Set*. 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 [15] gave a framework for MIS algorithms for the PRAM.
Luby's algorithm for MIS**:**
Step 1. Fix a random permutation $\pi : [n] \to [n]$ of vertices.
Step 2. Each vertex v adds itself to in MIS if $\pi(v) < \pi(u) \ \ \forall u \in N(v)$.
Step 4. Remove selected vertices and their neighbors.
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 working in the PRAM model that given a graph G of n vertices and m edges solves* MIS *in $O(\log n)$ depth and $O(m)$ work.*

Recently, Ghaffari & Uitto [10] showed that the local nature of the MIS problem can be efficiently exploited in the MPC model. Namely they proved the following.

Theorem 4. *There is an* MPC *algorithm, with memory per machine of $O(n^\epsilon)$ for any constant $\epsilon \in (0, 1)$, that, given a graph G of n vertices and m edges with probability at least $1 - \frac{1}{10n}$, solves* MIS *in $O(\sqrt{\log n} \cdot \log \log n)$.*

Theorem 4 is the best-known result for general graphs. Ghaffari, Grunau, Jin [8] showed that the MIS problem restricted, to some specific class of graphs (i.e. trees or graphs with bounded arboricity), has a randomized MPC algorithms working in $O(\log \log n)$ rounds in a strongly sublinear regime. Ghaffari, Kuhn, Uitto showed a (conditional) lower bound on 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.

Problem 3.3. *Fast Fourier Transform*
INSTANCE: Complex numbers $x_0, \ldots, x_{n-1}$.
QUESTION: Return the n complex numbers $X_0, \ldots, X_{n-1}$ where

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

Hajiaghayi et al. [11] 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 (3) $O(\frac{1}{\epsilon})$ rounds.*

4 Unconditional MPC Lower Bounds

In this section, we are going to present an approach to find unconditional lower bound for MPC problems which was introduced by Roughgarden et al. [17]. (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 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, 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 the 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, we describe how each bit of each machine determined by $\perp$-sum of its received signals.

Definition 5.

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 output of corresponding port. Now we understand how each input handles multiple signals, we describe the s-Shuffle more formally.

Definition 6. *An R rounds s-Shuffle, is a model with a set of V of machines such that each machine v has a round number $0 \leq r(v) \leq R+1$. Round 0 indicate input and receive input signals $x_1, x_2, \cdots, x_n$ and round $R+1$ indicate output and provide 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 define signals that machine u will send to all input of v according to output of u, i.e. machine u will send $\alpha_{u,v}(g(u))$ where $g(u)$ is output of machine u. Note that during a computation a machine will get many s-tuples.*

2. *We now define how the s-Shuffle computers. 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.*

Definition 7. *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 6. *A* 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 such that each machine has s entry.*

4.2 Polynomials.

In this section we describe how we can represent a s-Shuffle computation with a polynomial. First, the two basic bit operations **AND** and **OR** can be computed with polynomials:

- **AND**: $\prod x_i$
- **OR**: $1 - \prod(1 - x_i)$

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

Theorem 7. *Suppose that an s-Shuffle computation correctly computes the function $f : \{0,1\}^n \to \{0,1\}^k$ in r rounds. Then, there are k polynomials $\{p_i(x_1, \cdots, x_n)\}_{i=1}^k$ of degree at most s^r such that $p_i(x) = f(x)_i$ for all $i \in \{1, 2, \cdots, k\}$ and $x \in \{0,1\}^n$.*

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

The base of our induction is input machines. For each input x_i, we have a specific machine which its polynomial should be defined in this way:

- $p_{v,\mathbf{z}}(x_1, \cdots, x_n) = 1 - x_i$ 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 has at most $s^0 = 1$ degree. For induction step, lets 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 gets z_i from one of previous machines and $\perp$ from other machines. Therefore, for each machine u with $r(u) < r(v)$ and for each $z' \in \{0, 1, \perp\}^s$ 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 value other than $\perp$, this summation should not make any problem, i.e. for every input its value should be in $\{0, 1\}$.

2. $z_i = \perp$: In 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 degree of these polynomials are at most $s^{r(v)-1}$. In the end, for each $\mathbf{z}$, we set $p_{v,\mathbf{z}}$ equals 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

Theorem 8. *If a function cannot be represented by a polynomial with degree less than d, then we need at least $\lceil \log_s d \rceil$ rounds of s-Shuffle to compute this function.*

The Theorem 8 can be easily concluded from Theorem 7. Now, we want to use Theorem 8 to find lower bound for some functions. To obtain this, first we should find lower bound for degree of polynomials for these functions. We start with the most basic bit functions **AND**/**OR**. As we stated in previous section, the polynomial for computing **AND** of n elements is $\prod x_i$ and the polynomial for **OR** is $1 - \prod(1 - x_i)$. Now we can get lower bound for these functions in s-Shuffle.

Theorem 9 ([17]). *A s-Shuffle model that solves **AND/OR** function needs at least $\lceil \log_s n \rceil$ computation rounds.*

Now, we want to find lower bound for more complicated functions. Let's look at monotone graph properties. It has been showed that each monotone graph property need a decision tree with $\Omega(n^2)$ depth to be determined [16]. Also the decision tree depth of a polynomial with degree d is at most $2 \cdot d^4$[5]. Therefore, degree of a polynomial presenting a monotone graph property is at leas $\sqrt{n}$. Now we can conclude a lower bound for this type of functions.

Theorem 10 (Lower Bound for Monotone Graph Property [17]). *Every monotone graph property needs at least $\Omega(\log_s n)$ rounds of s-Shuffle computation.*

Another interesting problem that a lower bound has been found for it in this setting is connectivity in undirected graphs.

Theorem 11 (Lower Bound for Undirected Connectivity [17]). *Every s-Shuffle that computes the connectivity of undirected graphs needs at least $\lceil \log_s \binom{n}{2} \rceil$ computation rounds.*

We can find lower bound for many other functions by using this approach. Note that if s is a polynomial of n, i.e. $s = n^\epsilon$ where $\epsilon < 1$, these lower bounds that we presented are equal to $\log_s n = \frac{1}{\epsilon}$ which is not high enough. For example, if $s = n^\epsilon$ the best algorithm for connectivity of undirected graphs needs $\log n$ rounds to compute the answer if the number of machine is bounded. But, we are giving $\frac{1}{\epsilon}$ lower bound for this problem. Though, if we can have an unbounded width in s-Shuffle, we can solve any function with n inputs in $\lceil \log_s n \rceil$ rounds. We only need to build a tree for each possible input to verify it. We write this as the last theorem of this section.

Theorem 12 (Upper Bound in Unbounded s-Shuffle [17]). *Every function $f : \{0,1\}^n \to \{0,1\}^k$ can be computed with a s-Shuffle with unbounded width in $\lceil \log_s n \rceil$ rounds.*

5 Conditional MPC Lower Bounds

In this section, we are going to present an approach to find conditional lower bound for MPC problems which was introduced by Ghaffari et al. [9]. (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 1vs2-Cycle, we will prove 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 $\Omega_\epsilon(\log n)$ rounds. (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 13. *Assume the* 1vs2-Cycle *conjecture. Any strongly sublinear* MPC *for the 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 bound for other problems by making a reduction from 1vs2-Cycle or other problems that have a known conditional lower bound. But, we want to introduce a more advanced approach for finding conditional lower bounds based on Conjecture 1.

In the following, we are going to define the LOCAL model. Then, in Theorem 14, we show that how lower bounds in LOCAL model apply in MPC by using Conjecture 1. For proving this theorem, we use the maximal matching problem as an example.

The LOCAL model was defined by Linial [14] in 1987; hence, it is somewhat surprising it is used for lower bounds on the MPC model now. The following definition and theorem are from that paper.

Definition 8. *In LOCAL model, we have a graph that each node operates as a processor. In every round, each node sends data to its neighbors and receives data from them. This process will continue until every node has its output.*

Theorem 14. *If a graph problem needs r rounds of LOCAL computation, then the problem cannot be computed in strongly sublinear* MPC *within $o(\log r)$ rounds, unless the* 1vs2-Cycle *problem can be computed in strongly sublinear* MPC *in $o(\log n)$ rounds.*

Theorem 14 introduced and proved in [9] and we are going to go over their proof. They have an assumption called **component-stable** for MPC algorithms. An algorithm is **component-stable** if the output of every node is independent from topology of graph in other connected components. Now, we show why Theorem 14 holds by example. We know that computing a maximal matching needs $\Omega\left(\sqrt{\frac{\log n}{\log\log n}}\right)$ of LOCAL rounds. Thus, the number of rounds for this problem in strongly sublinear MPC based on Theorem 14 is at least $\Omega(\log\log n)$.

Theorem 15. *In strongly sublinear* MPC *model, there is no component-stable algorithm which can compute maximal matching in* $o(\log\log n)$ *rounds, unless the* 1vs2-Cycle *can be computed in strongly sublinear* MPC *in* $o(\log n)$ *rounds.*

For proving Theorem 15, we need to get familiar with an important concept in LOCAL model. In this model, we say that a problem has r **locality radius** if each node needs at least r rounds to find its output. The locality radius is important for us, because it shows that the output of node v is dependent to set of nodes with distance at most r from v. Now, consider that we have a graph and two selected nodes v and u from this graph such that these two nodes are either in different connected components or their distance is r. Now, if node u makes some changes in its area (e.g. changes id of its neighbors), we will have two different cases:

- If v and u are in a **same component**, the probability of changing the status of v (match to new node or switch between matched and unmatched) is high, because the changes we made around u are in locality radius of v.
- If v and u are in **different components**, the changes near u will not affect v, because we assume that algorithms are component-stable.

Now, we want to make a graph and select two nodes from this graph such that these two nodes are either in different connected components or their distance equals to d. To do this, we build the graph with n/d disjoint paths of length d. Then, we select two different nodes, v and u such that each of them are at the end of a path. If these nodes are on different end of a same path, their distance will be equal to d. Otherwise, they will be in different connected components. If there is an algorithm in strongly sublinear MPC that can determine whether these two nodes are in a same connected components or not, in $o(\log d)$ rounds, then there will be an algorithm that can solve 1vs2-Cycle in $o(\log n)$ rounds of strongly sublinear MPC, which has a contradiction with Conjecture 1.

Finally, we want to use this graph to complete the proof of Theorem 15. To this end, we set G equals to the graph described in previous paragraph with $d = \theta\left(\sqrt{\frac{\log n}{\log\log n}}\right)$ which is locality radius of maximal matching. Look at Figure 5 for understanding the topology of G and its two different cases. Now, we run maximal matching on graph G, and some modified versions of this graph. These versions differ from G only in neighborhood of u. Now, we run maximal matching in these graphs and check if state of v is same in G and other versions of G. If the state of v was the same for G and all other versions, we know that v and u are not in a same connected components. Otherwise, they are in a same connected component. If there is an algorithm that computes maximal matching in $o\left(\log\left(\sqrt{\frac{\log n}{\log\log n}}\right)\right) = o(\log\log n)$ rounds, then we can find whether u and v are in same component or not in $o(\log dist(u,v))$ which solves 1vs2-Cycle in $o(\log n)$ rounds. This contradicts Conjecture 1.

In the next theorem, we show all lower bounds that obtained with this approach in [9].

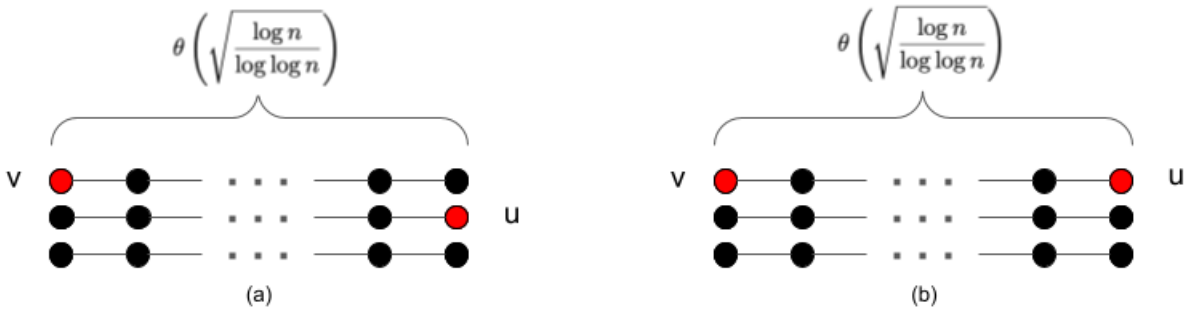

Figure 1: In picture (a), v and u are in different components. Therefore, modifications near u will not affect v. However, in picture (b), v and u has distance $\theta\left(\sqrt{\frac{\log n}{\log\log n}}\right)$. Thus, if we made some changes near u, we can see their effect on v.

Theorem 16 ([9])**.** *In strongly sublinear model, by assuming* 1vs2-Cycle *conjecture, these lower bounds proved for component-stable algorithms:*

- *Maximal Matching:* $\Omega(\log\log n)$
- *Sink-less Orientation:* $\Omega(\log\log\log n)$
- Δ*-vertex-coloring:* $\Omega(\log\log^* n)$

6 The AMPC Model

The AMPC model introduced in [4], extends the MPC model by allowing machines to have access to a shared memory. This model assumes that all messages sent in a single round are written to a distributed data storage, which all machines can read from within the next round. More specifically, the computation consists of rounds. In the i-th round, each machine can read data from a random access memory D_{i-1} and write to D_i (both D_{i-1} and D_i are common for all machines). Within a round, each machine can make up to $O(s)$ reads (henceforth called queries) and $O(s)$ writes and can perform arbitrary computation. Clearly, every MPC algorithm can be easily simulated in the AMPC model. However, the opposite direction is not usually the case. Furthermore, due to known simulations of PRAM algorithms by MPC, the AMPC model can also simulate existing PRAM algorithms. The key property of the AMPC model is that the queries a machine makes in each round may depend on the results of the previous queries it made in that same round, which is why the model is called *adaptive*. For example, if g is a function from X to X and for each $x \in X$, D_{i-1} stores a key-value pair $(x, g(x))$, then in round i a machine can compute $g^k(y)$ in a single round, provided that $k = O(S)$.

7 AMPC Power: 1vs2-Cycle Revisited.

In this section, we discuss the computation power of the AMPC model. For several fundamental problems, there are AMPC algorithms solving them with significantly lower round complexities

compared to their best-known MPC algorithms, which demonstrates the power of the AMPC model. [4] includes a number of such algorithms for some of the most fundamental graph problems, such as Graph Connectivity, Maximal Independent Set, Minimum Spanning Forest, Forest Connectivity, etc.
We further focus on a basic graph problem, namely the 1vs2-Cycle problem. In the MPC model, it is conjectured that solving this problem requires a logarithmic number of rounds. However, the following theorem shows that this conjecture does not hold in the Adaptive model.

Theorem 17 ([3]). *There is an* AMPC *algorithm solving the* 1vs2-Cycle *problem in* $O(1/\epsilon)$ *rounds w.h.p. using* $O(n^\epsilon)$ *space per machine and* $O(n)$ *total space.*

At first, we discuss the overview of their algorithm. In each round of the algorithm, we sample each vertex with probability $n^{-\epsilon/2}$. Then, we contract the original graph to the samples by replacing the paths between sampled vertices with single edges. To do so, we traverse the cycle in both directions for each sampled vertex until we hit another sampled vertex, which can be done in a single round using the adaptivity of the model. In each round, with high probability, the number of vertices shrinks by a factor of $n^{\epsilon/2}$. Therefore, after $O(1/\epsilon)$ rounds, the number of remaining vertices and edges is reduced to $O(n^\epsilon)$. Hence, the graph fits in the memory of a single machine, and we can solve the remaining problem in a single round.

The algorithm is as follows:

Algorithm 1 Shrink$(G = (V, E), \epsilon, t)$

for $i = 0, \ldots, t$ **do**
  $V' \leftarrow$ a subset of $V(G)$ s.t. each vertex is included independently with probability $n^{-\epsilon/2}$
  $E' \leftarrow \emptyset$
  for $v \in V'$ **do**
    $l_v \leftarrow$ first sampled vertex that we reach traversing the graph starting with $(v, nei^1_G(v))$
    $r_v \leftarrow$ first sampled vertex that we reach traversing the graph starting with $(v, nei^2_G(v))$
    $E' \leftarrow E' \cup \{(v, l_v), (v, r_v)\}$
  $G \leftarrow G'(V', E')$
Return G

Algorithm 2 1vs2-Cycle on $G(V, E)$

$G' \leftarrow$ Shrink$(G, \epsilon, O(1/\epsilon))$
Solve the 1vs2-Cycle problem on G' using a single machine ▷ $|V(G')| \in O(n^\epsilon)$ w.h.p.

Each iteration of the outer loop in Shrink is a single AMPC round, and the correctness of this algorithm is a result of combining the following lemmas.

Lemma 1 ([4]). *Let* G *be a graph consisting of cycles, and let* N *be the initial number of vertices in* G*. Consider a cycle with size* $k = \Omega(N^\epsilon)$ *in some iteration of the loop of Shrink* $(G, \epsilon, O(1/\epsilon))$*. The size of this cycle shrinks by at least a factor of* $N^{\epsilon/2}$ *after this iteration w.h.p.*

Lemma 2 ([4]). *Let* G *be a* N*-vertex graph consisting of cycles and let* $G' = Shrink(G, \epsilon, O(1/\epsilon))$*. Then* G' *can be obtained from* G *by contracting edges, and the length of each cycle in* G' *is* $O(n^\epsilon)$*.*

Lemma 3 ([4]). *In each round, the total communication of each machine is $O(N^\epsilon)$ w.h.p., where N is the initial number of vertices in G.*

8 Unconditional AMPC Lower Bounds.

In [6] polynomial method introduced in [17] is generalized to be used in the AMPC model. In regard to this matter, the following extension of the degree is introduced.

Definition 9. *for a partial Boolean function $g : \Delta \to \{0,1\}$, $deg_{partial}$ is defined as below:*

$$deg_{partial}(g) = \min\{deg(p) | p(x) = g(x) \text{ for all } x \in \Delta\}$$

The next theorem reduces lower-bounding the deterministic AMPC round complexity of g to lower-bounding $deg_{partial}(g)$:

Theorem 18 ([6]). *Any deterministic* AMPC *algorithm that computes a partial Boolean function $g : \Delta \to \{0,1\}$ requires $\frac{1}{2}\log_S deg_{partial}(g)$ rounds. In particular, if g is a total Boolean function, then any such algorithm requires $\frac{1}{2}\log_S deg(g)$ rounds.*

Considering randomized AMPC algorithms, they say a Boolean function $g : \Delta \to \{0,1\}$ is approximately represented by a polynomial p if $|p(x) - g(x)| \leq \frac{1}{3}$ for any $x \in \Delta$; and introduce $\tilde{deg}_{partial}$.

Definition 10. *for a partial Boolean function $g : \Delta \to \{0,1\}$, $\tilde{deg}_{partial}$ is defined as below:*

$$\tilde{deg}_{partial}(g) = \min\{deg(p) | p \text{ approximately represents } g\}$$

The following theorem reduces lower-bounding the randomized AMPC round complexity of g to lower-bounding $\tilde{deg}_{partial}(g)$:

Theorem 19 ([6]). *Any randomized* AMPC *algorithm that computes a partial Boolean function $g : \Delta \to \{0,1\}$ with error at most $\frac{1}{3}$ requires $\frac{1}{2}\log_S \tilde{deg}_{partial}(g)$ rounds. In particular, if g is a total Boolean function, then any such algorithm requires $\frac{1}{2}\log_S \tilde{deg}(g)$ rounds.*

The following theorems are examples of lower bounds in the AMPC model for problems such as 1vs2-Cycle drawn from previously mentioned theorems.

Theorem 20 (Unconditional AMPC Round Lower Bound For 1vs2-Cycle [6]). *Any deterministic* AMPC *algorithm for* 1vs2-Cycle *on n vertices or any randomized* AMPC *algorithm that computes* 1vs2-Cycle *with error at most $\frac{1}{3}$ requires $\frac{1}{2}\log_S \Theta(n) - 1$ rounds. In particular for $S = n^\epsilon$ any such algorithm requires $\Omega(1/\epsilon)$ rounds.*

Proof. We prove the theorem using a reduction from Parity to 1vs2-Cycle. From an instance $x = \{x_1, \ldots, x_N\} \in \{0,1\}^N$ of Parity, we construct the graph $G(x)$ as follows: For any x_i we add vertices v_i^1 and v_i^2, and for any $i = 1, \ldots, N$, if $x_i = 0$, we add edges $(v_i^1, v_{(i \mod N)+1}^1)$ and $(v_i^2, v_{(i \mod N)+1}^2)$; otherwise, we add edges $(v_i^1, v_{(i \mod N)+1}^2)$ and $(v_i^2, v_{(i \mod N)+1}^1)$. See Figure 2 for more illustration. It is clear that for any $i = 1, \ldots, N-1$, after adding its corresponding edges, we have two distinct paths of length i on vertices $\{v_1^1, v_1^2, \ldots, v_{i+1}^1, v_{i+1}^2\}$, and for any $j \in \{1, \ldots, i+1\}$, v_j^1 and v_j^2 are in different paths. The last pair of edges either connect these paths and form a cycle

of length $2N$ or connect the endpoints of each of these paths and form two cycles of length N. If we partition the vertices to $\{v_1^1, \ldots, v_N^1\}$ and $\{v_1^2, \ldots, v_N^2\}$, and follow the path starting from v_1^1 according to the order of incoming edges, it is clear that when edges corresponding i are added we change partition iff $x_i = 1$. Thus, this path reaches v_1^2 iff the number of times we change partition is odd, which means $Parity(x) = 1$. Thus, this graph is a cycle of length $2N$ iff $Parity(x) = 1$. This reduction can be done using a single round of AMPC computation. For each $i = 1, \ldots, N$, the machine that reads the bit x_i adds the edges corresponding to i according to the value of x_i by writing them in the shared memory. Hence, if any deterministic/randomized AMPC algorithm solves the problem of 1vs2-Cycle on a graph of order N in $f(N)$ rounds, then we can compute the Parity: $\{0,1\}^n \to \{0,1\}$ in $f(2N) + 1$ rounds. It is well-known that $deg(Parity) = N$,and $\tilde{deg}(Parity) = \Theta(N)$. Using Theorems 18 and 19, the proof is complete. □

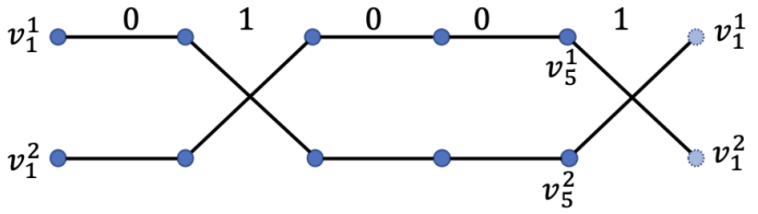

Figure 2: Reduction from a Parity instance $x = 01001$ to a 1vs2-Cycle instance with 10 vertices.

Theorem 21 (Unconditional AMPC Round Lower Bound For 1vsk-Cycle [6])**.** *Any deterministic* AMPC *algorithm for* 1vsk-Cycle *on n vertices or any randomized* AMPC *algorithm that computes* 1vsk-Cycle *with error at most $\frac{1}{3}$ requires $\frac{1}{2}\log_S \Theta(\frac{n}{k^2}) - 1$ rounds. In particular, if $k = O(n^\delta)$ for $\delta \in (0, \frac{1}{2})$ and $S = n^\epsilon$ for $\epsilon \in (0,1)$, then any such algorithm requires $\Omega((1-2\delta)/\epsilon)$ rounds.*

Here we give an overview of its proof.

Proof. Similar to the proof of Theorem 20, we reduce Parity to 1vsk-Cycle. Given an instance $x = \{x_1, \ldots, x_N\} \in \{0,1\}^N$ of Parity, we construct the graph $G(x)$ just like we did in the proof of the Theorem 20. Then, like Figure 2, we construct the graph $G'(x)$ using k copy of $G(x)$ and $2k-2$ undirected paths. $G'(x)$ consists of k cycles if and only if $Parity(x) = 0$; otherwise, it is one cycle passing through all the vertices in G'. Hence, the lower bound of computing the Parity function proves implies our lower bound. □

9 Future Directions.

Despite the recent interest in both MPC and AMPC models, there are many fundamental problems that have not yet been understood with the expected level of detail. Examples of such problems are 2-SAT and directed S-T connectivity. These problems are related in the sense that solving 2-SAT is usually done by reducing it to directed $S-T$ connectivity. On the other hand, directed $S-T$ connectivity seems a natural environment in which the AMPC assumption of *adaptivity* should be superior to the other considered models (e.g. PRAM). This perspective can be inspiring to investigate the mentioned problems in the context of massively parallel computations.

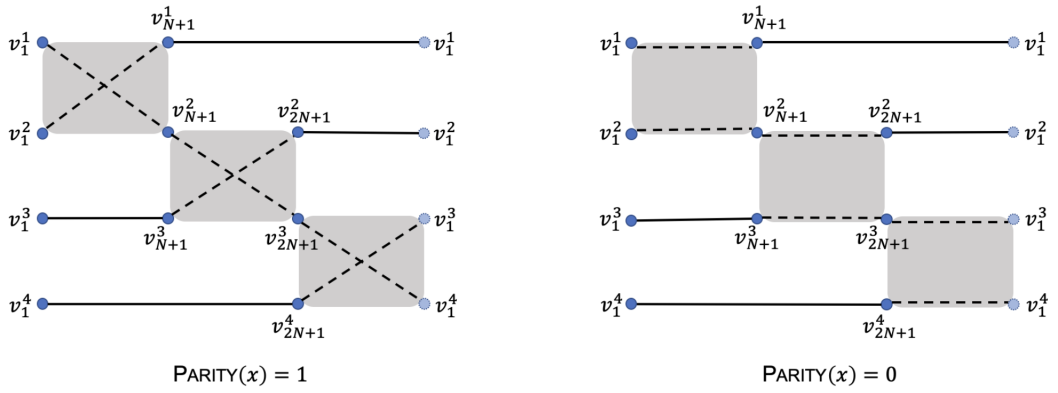

Figure 3: Reduction from a Parity instance to a 1vsk-Cycle-Cycle instance with $k = 4$.

References

[1] A. Andoni, Z. Song, C. Stein, Z. Wang, and P. Zhong. Parallel graph connectivity in log diameter rounds. In M. Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 674–685. IEEE Computer Society, 2018.
`https://doi.org/10.1109/FOCS.2018.00070`.

[2] P. Beame, P. Koutris, and D. Suciu. Communication steps for parallel query processing. *Journal of the Assocation of Computing Machinery*, 64(6):40:1–40:58, 2017.
`https://doi.org/10.1145/3125644`.

[3] S. Behnezhad, L. Dhulipala, H. Esfandiari, J. Lacki, and V. S. Mirrokni. Near-optimal massively parallel graph connectivity. In D. Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 1615–1636. IEEE Computer Society, 2019.
`https://doi.org/10.1109/FOCS.2019.00095`.

[4] S. Behnezhad, L. Dhulipala, H. Esfandiari, J. Lacki, W. Schudy, and V. S. Mirrokni. Massively parallel computation via remote memory access, 2019.
`http://arxiv.org/abs/1905.07533`.

[5] H. Buhrman and R. de Wolf. Complexity measures and decision tree complexity: a survey. *Theor. Comput. Sci.*, 288(1):21–43, 2002.

[6] M. Charikar, W. Ma, and L. Tan. Unconditional lower bounds for adaptive massively parallel computation. In *SPAA*, pages 141–151. ACM, 2020.

[7] S. A. Cook, C. Dwork, and R. Reischuk. Upper and lower time bounds for parallel random access machines without simultaneous writes. *SIAM J. Comput.*, 15(1):87–97, 1986.
`https://doi.org/10.1137/0215006`.

[8] M. Ghaffari, C. Grunau, and C. Jin. Improved MPC algorithms for MIS, matching, and coloring on trees and beyond. In H. Attiya, editor, *34th International Symposium on Distributed Computing, DISC 2020, October 12-16, 2020, Virtual Conference*, volume 179 of *LIPIcs*, pages 34:1–34:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. https://doi.org/10.4230/LIPIcs.DISC.2020.34.

[9] M. Ghaffari, F. Kuhn, and J. Uitto. Conditional hardness results for massively parallel computation from distributed lower bounds. In D. Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 1650–1663. IEEE Computer Society, 2019. https://doi.org/10.1109/FOCS.2019.00097.

[10] M. Ghaffari and J. Uitto. Sparsifying distributed algorithms with ramifications in massively parallel computation and centralized local computation. In T. M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1636–1653. SIAM, 2019. https://doi.org/10.1137/1.9781611975482.99.

[11] M. Hajiaghayi, H. Saleh, S. Seddighin, and X. Sun. String matching with wildcards in the massively parallel computation model. In K. Agrawal and Y. Azar, editors, *SPAA '21: 33rd ACM Symposium on Parallelism in Algorithms and Architectures, Virtual Event, USA, 6-8 July, 2021*, pages 275–284. ACM, 2021. https://doi.org/10.1145/3409964.3461793.

[12] R. Karp. A survey of parallel algorithms for shared-memory machines, 1988. http://www2.eecs.berkeley.edu/Pubs/TechRpts/1988/CSD-88-408.pdf.

[13] M. Li and Y. Yesha. New lower bounds for parallel computation. *J. ACM*, 36(3):671–680, 1989. https://doi.org/10.1145/65950.65959.

[14] N. Linial. Distributive graph algorithms-global solutions from local data. In *FOCS*, pages 331–335. IEEE Computer Society, 1987. https://ieeexplore.ieee.org/document/4568287.

[15] M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM J. Comput.*, 15(4):1036–1053, 1986. https://doi.org/10.1137/0215074.

[16] A. L. Rosenberg. On the time required to recognize properties of graphs: a problem. *SIGACT News*, 5(4):15–16, 1973.

[17] T. Roughgarden, S. Vassilvitskii, and J. R. Wang. Shuffles and circuits (on lower bounds for modern parallel computation). *J. ACM*, 65(6):41:1–41:24, 2018. https://dl.acm.org/doi/10.1145/3232536.