Lower bounds in MPC and AMPC models

Samira Goudarzi, Peyman Jabbarzade, Jan Olkowski

March 2022

1 The MPC model

The Massively Parallel Computation model is a modern theoretical approach for developing parallel / distributed algorithms. Its main feature is the fact the size of input data outstrips the memory and communication availability of a single machine. Therefore, the model assumes the existence of multiple machines for computation. As such, the MPC model is most often defined by three parameters — the input data size N, the number of machines M available for computation, and the memory size per machine of 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

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

which can be interpreted that the most interesting scenario is when the number of available resources for the algorithm ($M \cdot S$) is asymptotically equal to the number of the input data. However, line 1 is by no means a formal assumption of the model and there might be problems (e.g. 1vs2-Cycle) in which results deriving from this rule are still desirable.

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/receive at most S words, however, each of these words can be addressed to / received from different machines. The two main parameters that are investigated in this model are:
1) the number of communication rounds it takes for an algorithm to solve a problem, often called *running time*. Here, it is worth noting 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.
2) the size of local memory S. Problems are easier to solve with larger S. In the extreme when $S >> N$, 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 $\epsilon < 1$.
Initial data distribution. The whole input data is split across the M machines *arbitrarily*. For example, each edge of a graph is stored arbitrarily on some machine.

1.1 General view: PRAM model

The precursor of the two mentioned models is the Parallel Random Access Memory model (PRAM). In this model, we have N processes that work on a shared Random Access Memory. Processes work in instructions. Each instruction consists of three phases: an optional read from the shared memory, a single computation instruction, and an optional write to the shared memory. It is assumed that executing a single instruction takes a single unit of time. Therefore, *the running time* (often called also *a depth*) of an algorithm is the longest time of terminating a single processor. Obviously, in the above model can occur access conflicts, e.g. when two processes want to write the same cell of memory at the writing phase of a single instruction. This is dealt with in several ways, including:

- **EREW-PRAM** The exclusive-read exclusive-write PRAM requires that no two processors simultaneously access any given memory cell.

- **CREW-PRAM** The concurrent-read exclusive-write PRAM permits simultaneous reads of the same memory cell, but only one processor may attempt to write to the cell.
- **CRCW-PRAM** The concurrent-read concurrent-write PRAM permits simultaneous reads and writes to the same memory cell. Some method of arbitrating simultaneous writes to the same cell is required. For example, in the *ARBITRARY* version any one of the writes succeeds.

Fact 1. *Any PRAM algorithm working in depth t can be simulated by an MPC algorithm in $O(t)$ rounds and with strongly sublinear memory per machine.*

2 Overview of some fundamental problems in MPC model.

In this section, we shortly discuss how the MPC model is superior to the PRAM model. We review the state-of-art algorithms for some fundamental graphs problems providing an intuition on the advantages of MPC over PRAM.

2.1 Connectivity.

Definition 1 (Connectivity)**.** *Given an undirected graph $G = (V, E)$, a connected component is a subgraph in which any two vertices are connected by a path. A solution to the connectivity instance 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 optimal solutions to the Connectivity problem have the running time $O(\log D + \log_{m/n} n)$ in both models: MPC and PRAM, but interestingly, the algorithm working in MPC model was proposed before PRAM solution. This result are summarized in the following theorems.

Theorem 1 ([**DBLP:conf/focs/BehnezhadDELM19**])**.** *There is a strongly sublinear MPC algorithm with $O(m)$ total space that given a graph with diameter D, identifies its connected components in $O(\log D)$ rounds if $D \geq \log n$, and takes $O(\log \log_{m/n} n)$ rounds otherwise. The algorithm is randomized, succeeds with high probability, and does not require prior knowledge of D.*

Theorem 2 ([**DBLP:conf/focs/BehnezhadDELM19**])**.** *There is a CRCW PRAM algorithm that given a graph with diameter D, identifies its connected components in $O(\log D + \log \log_{m/n} n)$ depth and $O((m + n)(\log D + \log \log_{m/n} n))$ work. The algorithm is randomized, succeeds with high probability and does not require prior knowledge of D.*

We will not provide proof of the two above results. Instead, we will give a short overview of the slightly weaker result.

Theorem 3 ([**DBLP:conf/focs/AndoniSSWZ18**])**.** *There is a strongly sublinear MPC algorithm with $O(m)$ total space that given a graph with diameter D, identifies its connected components in $O(\log D \cdot \log \log_{m/n} n)$. The algorithm is randomized, succeeds with high probability, and does not require prior knowledge of D.*

This is what [**DBLP:conf/focs/BehnezhadDELM19**] writes about the ideas which lead to Theorem 3

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 $d >> \log n$. 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 $O(\log D \cdot \log\log n)$.

2.2 Maximal Independent Set.

Definition 2 (Maximal Independent Set)**.** *For a graph $G = (V, E)$ an independent set S maximal independent set if for $v \in V$, one of the following is true:*
a) $v \in S$
b) $N(v) \cap S \neq \emptyset$ where $N(v)$ denotes the neighbors of v.
In the MIS problem the goal is to output any maximal independent set S.

First, we state a general framework, proposed by Luby in [**DBLP:journals/siamcomp/Luby86**], which at the same time is the best-known algorithm (i.e. has the lowest depth) for MIS in PRAM model.
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 [**DBLP:journals/siamcomp/Luby86**].

Theorem 4 ([**DBLP:journals/siamcomp/Luby86**])**.** *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 and Uitto in [**DBLP:conf/soda/GhaffariU19**] showed that the local nature of MIS problem can be efficiently exploited in the MPC model. Namely. they proved the following.

Theorem 5. *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 $O(\sqrt{\log n} \cdot \log\log n)$.*

The above theorem is the best-known result for general graphs. If restricted to some specific class of graphs, i.e. trees or graphs with bounded arboricity, then in [**DBLP:conf/wdag/GhaffariGJ20**] authors show randomized MPC algorithms working in $O(\log\log n)$ rounds in a strongly sublinear regime. The best known (conditional) lower bound on MIS problem in the MPC model states that $\Omega(\log\log n)$ rounds are necessary in the case for some graph, see [**DBLP:conf/focs/GhaffariKU19**].

2.3 Fast Fourier Transform.

Definition 3 (Fast Fourier Transform)**.** *Let $x_0, \ldots, x_{n-1}$ be complex numbers. In Fast Fourier Transform the goal is to compute n complex values $X_0, \ldots, X_{n-1}$ defined as below:*

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

Utilizing butterfly graphs in the MPC model gives the following result.

Theorem 6. *For any $\epsilon > 0$, there exists a deterministic algorithm working in the MPC model with the local memory $O(n^\epsilon)$ and the total memory $O(npolylogn)$ that computes FFT in $O(\frac{1}{\epsilon})$ rounds.*

This should be contrasted with the folklore result in the PRAM model.

Theorem 7. *In the PRAM model, there exists an algorithm that computes FFT in $O(\log n)$ depth and uses in $O(n\ polylog(n))$ work.*

3 Unconditional MPC lower bounds.

In this section, we are going to present an approach to find lower bound for MPC problems which introduced in [**DBLP:journals/jacm/RoughgardenVW18**]. To do this, we first introduce an abstract model of MPC named S-Shuffle. Then, we show that computations in this model can be represented by polynomials. Finally, we use lower bound on polynomial degree of boolean functions to achieve lower bound on number of rounds in MPC model. Although these lower bounds are not tight, they are important because they are unconditional.

3.1 Generalized MPC: s-Shuffle model.

The s-Shuffle model, is an abstract model for MPC. In this model, we have multiple gates. Each gate, has S inputs and located in one of R levels. Each input of each gate, get a signal from all other gates in lower levels. These signals should be 0, 1, or $\perp$. For each input, at most one of its signal can be other than $\perp$. In following, we describe that how each bit of each gate determined by $\perp$-sum of its received signals.

Definition 4 ($\perp$-sum [**DBLP:journals/jacm/RoughgardenVW18**])**.** *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*

For each port of every gate, $\perp$-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 5 (s-Shuffle [**DBLP:journals/jacm/RoughgardenVW18**])**.** *A R rounds s-Shuffle, is a model with set V of gates such that each gate 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 $y_1, y_2, \cdots, y_k$. 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 gate u will send to all input of v according to output of u, i.e. gate u will send $\alpha_{u,v}(g(u))$ where $g(u)$ is output of gate u. Then, for each entry of v, its value will be determined by $\perp$-sum of all signals received on that entry.*

The last definition we need for this part is **width** which is the maximum number of machines in a round other than rounds 0 and $R+1$. Next we are going to show the relation between s-Shuffle and MPC model.

Theorem 8 (MPC and s-Shuffle Relation [**DBLP:journals/jacm/RoughgardenVW18**])**.** *A MPC computation that runs on M machines with space S in R rounds can be simulated with a s-Shuffle instance with width M in R rounds such that each gate has S entry.*

3.2 Polynomials.

In this section we describe that 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 9 (Round-Efficient Shuffles Are Low-Degree Polynomials [**DBLP:journals/jacm/RoughgardenVW18**]). *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 gate 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 gate 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 gates. For each input x_i, we have a specific gate 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 gates 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 gate v should gets z_i from one of previous gates and $\perp$ from other gates. Therefore, for each gate 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 $s^{r(v)}$. □

3.3 Lower bound in MPC.

Theorem 10 (Lower Bound in s-Shuffle [**DBLP:journals/jacm/RoughgardenVW18**]). *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 10 can be easily concluded from Theorem 9. Now, we want to use Theorem 10 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 11 ([**DBLP:journals/jacm/RoughgardenVW18**]). *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 [**DBLP:journals/sigact/Rosenberg73**]. Also the decision tree depth of a polynomial with degree d is at most $2 \cdot d^4$[**DBLP:journals/tcs/BuhrmanW02**]. 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 12 (Lower Bound for Monotone Graph Property [**DBLP:journals/jacm/RoughgardenVW18**]). *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 13 (Lower Bound for Undirected Connectivity [**DBLP:journals/jacm/RoughgardenVW18**])**.** *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 14 (Upper Bound in Unbounded s-Shuffle [**DBLP:journals/jacm/RoughgardenVW18**])**.** *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.*

4 Conditional MPC lower bounds.

In this section, we are going to introduce a conjecture about 1vs2-Cycle problem and use it to find some lower bounds in strongly sublinear MPC. These lower bounds hold if the conjecture is true. Let's go over 1vs2-Cycle and its conjecture. Then, we obtain other lower bounds from this conjecture.

Definition 6 (1vs2-Cycle)**.** *Given an undirected graph* $G = (V, E)$ *of size* n*, the problem is to detect wether the graph consists of one cycle, or two cycles.*

The following conjecture is widely believed in the MPC literature about lower bound of 1vs2-Cycle.

Conjecture 1 (Logarithmic lower bound for 1vs2-Cycle)**.** *Any MPC algorithm for the 1v2-Cycle problem using machines with I/O capacity* $s = n^\epsilon$ *requires* $\Omega_\epsilon(logn)$ *rounds.*

In fact, such a 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 $n/2$. Note that this conjecture holds in strongly sublinear MPC models where the space of each machine is polynomial of n, i.e. $s = n^\epsilon$ where $\epsilon < 1$.

Based on this conjecture, a number of works had shown conditional hardness results for a variety of problems in MPC. For example, we make a reduction from 1vs2-cycle to undirected connectivity problem to find conditional lower bound for connectivity problem.

Theorem 15. *By assuming 1vs2-cycle conjecture, computing undirected connectivity in strongly sublinear MPC needs at least* $\Omega(\log n)$ *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 has contradiction. □

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 LOCAL model. Then, in Theorem 16, we show that how lower bounds in LOCAL model apply in MPC by using Conjecture 1. For proving this theorem, we use problem maximal matching as an example.

Definition 7 (LOCAL model [**DBLP:conf/focs/Linial87**])**.** *In LOCAL model, we have a graph that each node operates as a processor. In every round, each node sends data to its neighbours and receives data from them. This process will continue until every node has its output.*

Theorem 16 (Conditional Lower Bound in MPC [**DBLP:conf/focs/GhaffariKU19**])**.** *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 16 introduced and proved in [**DBLP:conf/focs/GhaffariKU19**] and we are going to go over their proof. They have a 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 16 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 16 is at least $\Omega(\log\log n)$.

Theorem 17 (Conditional Lower Bound for Maximal Matching [**DBLP:conf/focs/GhaffariKU19**])**.** *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 17, 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 dependant 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 neighbours), 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.

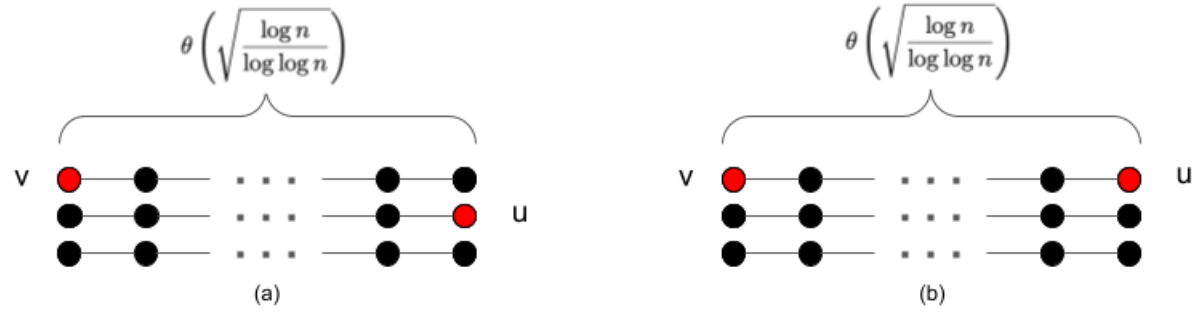

Figure 1: In picture (a), v and u are in different components. Therefore, modifications near u will not affect v. However, in picture (b), v and u has distance $\theta\left(\sqrt{\frac{\log n}{\log\log n}}\right)$. Thus, if we made some changes near u, we can see their effect on v.

Finally, we want to use this graph to complete the proof of Theorem 17. 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 4 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 neighbourhood 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 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 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 has a contradiction with Conjecture 1 and Theorem 17 proved.

In the next theorem, we show all lower bounds that obtained with this approach in [**DBLP:conf/focs/GhaffariKU19**].

Theorem 18 ([**DBLP:conf/focs/GhaffariKU19**])**.** *In strongly sublinear model, by assuming 1vs2-Cycle conjecture, these lower bounds proved for component-stable algorithms:*

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

5 AMPC model

The AMPC model introduced in [**DBLP:journals/corr/abs-1905-07533**], 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 $k = O(S)$.

6 AMPC power: 1v2-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. [**DBLP:journals/corr/abs-1905-07533**] 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 19 ([**DBLP:conf/focs/BehnezhadDELM19**])**.** *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 1v2-Cycle(G(V, E))

$G' \leftarrow$ Shrink($G, \epsilon, O(1/\epsilon)$)
Solve the 1v2-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 ([**DBLP:journals/corr/abs-1905-07533**]). *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 ([**DBLP:journals/corr/abs-1905-07533**]). *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 ([**DBLP:journals/corr/abs-1905-07533**]). *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.*

7 Unconditional AMPC lower bounds.

In [**DBLP:conf/spaa/CharikarMT20**] polynomial method introduced in [**DBLP:journals/jacm/RoughgardenVW18**] is generalized to be used in the AMPC model. In regard to this matter, the following extension of the degree is introduced.

Definition 8. *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 20 ([**DBLP:conf/spaa/CharikarMT20**]). *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 9. *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 21 ([**DBLP:conf/spaa/CharikarMT20**])**.** *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 22 (Unconditional AMPC round lower bound for 1v2-Cycle [**DBLP:conf/spaa/CharikarMT20**])**.** *Any deterministic AMPC algorithm for 1v2-Cycle on n vertices or any randomized AMPC algorithm that computes 1v2-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 1v2-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 $(v_i^1, v^1_{(i \mod N)+1})$ and $(v_i^2, v^2_{(i \mod N)+1})$; otherwise, we add edges $(v_i^1, v^2_{(i \mod N)+1})$ and $(v_i^2, v^1_{(i \mod N)+1})$. See Figure 1 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 1v2-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 20 and 21, the proof is complete. □

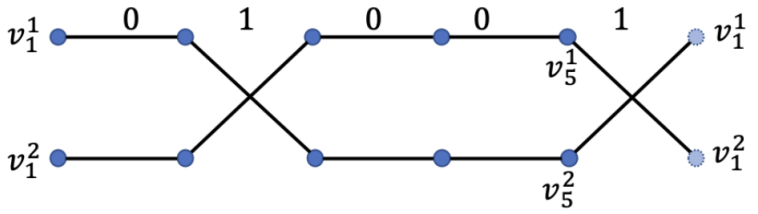

Figure 2: Reduction from a Parity instance $x = 01001$ to a 1v2-Cycle instance with 10 vertices.

Theorem 23 (Unconditional AMPC round lower bound for 1vk-Cycle [**DBLP:conf/spaa/CharikarMT20**])**.** *Any deterministic AMPC algorithm for 1vk-Cycle on n vertices or any randomized AMPC algorithm that computes 1vk-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 22, we reduce Parity to 1vk-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 22. 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. □

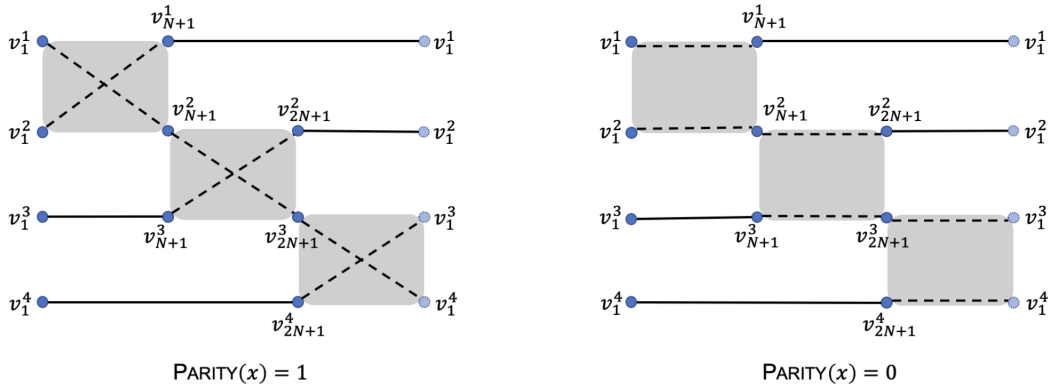

Figure 3: Reduction from a Parity instance to a 1vk-Cycle instance with $k = 4$.

8 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.