Round Compression for Parallel Matching Algorithms

Artur Czumaj*
University of Warwick
United Kingdom
A.Czumaj@warwick.ac.uk

Jakub Łącki
Google Research
New York, NY, USA
jlacki@google.com

Aleksander Mądry†
MIT
Cambridge, MA, USA
madry@mit.edu

Slobodan Mitrović‡
EPFL
Lausanne, Switzerland
slobodan.mitrovic@epfl.ch

Krzysztof Onak
IBM Research
Yorktown Heights, NY, USA
konak@us.ibm.com

Piotr Sankowski§
University of Warsaw
Warszawa, Poland
sank@mimuw.edu.pl

ABSTRACT

For over a decade now we have been witnessing the success of *massive parallel computation* (MPC) frameworks, such as MapReduce, Hadoop, Dryad, or Spark. One of the reasons for their success is the fact that these frameworks are able to accurately capture the nature of large-scale computation. In particular, compared to the classic distributed algorithms or PRAM models, these frameworks allow for much more local computation. The fundamental question that arises in this context is though: can we leverage this additional power to obtain even faster parallel algorithms?

A prominent example here is the *maximum matching* problem—one of the most classic graph problems. It is well known that in the PRAM model one can compute a 2-approximate maximum matching in $O(\log n)$ rounds. However, the exact complexity of this problem in the MPC framework is still far from understood. Lattanzi et al. (SPAA 2011) showed that if each machine has $n^{1+\Omega(1)}$ memory, this problem can also be solved 2-approximately in a constant number of rounds. These techniques, as well as the approaches developed in the follow up work, seem though to get stuck in a fundamental way at roughly $O(\log n)$ rounds once we enter the (at most) near-linear memory regime. It is thus entirely possible that in this regime, which captures in particular the case of sparse graph computations, the best MPC round complexity matches what one can already get in the PRAM model, without the need to take advantage of the extra local computation power.

In this paper, we finally refute that possibility. That is, we break the above $O(\log n)$ round complexity bound even in the case of *slightly sublinear* memory per machine. In fact, our improvement here is *almost exponential*: we are able to deliver a $(2+\epsilon)$-approximate maximum matching, for any fixed constant $\epsilon > 0$, in $O\left((\log \log n)^2\right)$ rounds.

To establish our result we need to deviate from the previous work in two important ways that are crucial for exploiting the power of the MPC model, as compared to the PRAM model. Firstly, we use *vertex–based* graph partitioning, instead of the edge–based approaches that were utilized so far. Secondly, we develop a technique of *round compression*. This technique enables one to take a (distributed) algorithm that computes an $O(1)$-approximation of maximum matching in $O(\log n)$ *independent* PRAM phases and implement a super-constant number of these phases in only a constant number of MPC rounds.

CCS CONCEPTS

• **Theory of computation → MapReduce algorithms**; **Massively parallel algorithms**;

KEYWORDS

maximum matching, vertex partitioning, round compression

ACM Reference Format:
Artur Czumaj, Jakub Łącki, Aleksander Mądry, Slobodan Mitrović, Krzysztof Onak, and Piotr Sankowski. 2018. Round Compression for Parallel Matching Algorithms. In *Proceedings of 50th Annual ACM SIGACT Symposium on the Theory of Computing (STOC'18).* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3188745.3188764

1 INTRODUCTION

Over the last decade, massive parallelism became a major paradigm in computing, and we have witnessed the deployment of a number of very successful massively parallel computation frameworks, such as MapReduce [16, 17], Hadoop [38], Dryad [25], or Spark [39]. This paradigm and the corresponding models of computation are rather different from classical parallel algorithms models considered widely in literature, such as the PRAM model. In particular, in this paper, we study the *Massive Parallel Computation (MPC)* model (also known as Massively Parallel Communication model) that was abstracted out of capabilities of existing systems, starting with the work of Karloff, Suri, and Vassilvitskii [3, 8, 9, 21, 29]. The main

*Supported in part by the Centre for Discrete Mathematics and its Applications (DIMAP), Royal Society International Exchanges Scheme 2013/R1, IBM Faculty Award, and the EPSRC award EP/N011163/1.
†Supported in part by an Alfred P. Sloan Research Fellowship, Google Research Award, and the NSF grant CCF-1553428.
‡Supported in part by the Swiss NSF grant P1ELP2_161820.
§Supported in part by grant NCN2014/13/B/ST6/00770 of Polish National Science Center and ERC StG grant TOTAL no. 677651.

STOC'18, June 25–29, 2018, Los Angeles, CA, USA

ACM ISBN 978-1-4503-5559-9/18/06…$15.00
https://doi.org/10.1145/3188745.3188764

difference between this model and the PRAM model is that the MPC model allows for much more (in principle, unbounded) local computation. This enables it to capture a more "coarse–grained," and thus, potentially, more meaningful aspect of parallelism. It is often possible to simulate one clock step of PRAM in a constant number of rounds on MPC [21, 29]. This implies that algorithms for the PRAM model usually give rise to MPC algorithms without incurring any asymptotic blow up in the number of parallel rounds. As a result, a vast body of work on PRAM algorithms naturally translates to the new model.

It is thus natural to wonder: Are the MPC parallel round bounds "inherited" from the PRAM model tight? In particular, which problems can be solved in significantly *smaller* number of MPC rounds than what the lower bounds established for the PRAM model suggest?

It is not hard to come up with an example of a problem for which indeed the MPC parallel round number is much smaller than its PRAM round complexity. For instance, computing the parity of n Boolean values takes only $O(1)$ parallel rounds in the MPC model when space per machine is $n^{\Omega(1)}$, while on PRAM it provably requires $\Omega(\log n/\log\log n)$ time [7] (as long as the total number of processors is polynomial). However, the answer is typically less obvious for other problems. This is particularly the case for graph problems, whose study in a variant of the MPC model was initiated already by Karloff et al. [29].

In this paper, we focus on one such problem, which is also one of the most central graph problems both in sequential and parallel computations: maximum matching. Maximum matchings have been the cornerstone of algorithmic research since 1950s and their study inspired many important ideas, including the complexity class P [18]. In the PRAM model we can compute $(1+\epsilon)$-approximate matching in $O(\log n)$ rounds [32] using randomization. Deterministically, a $(2+\epsilon)$-approximation can be computed in $O\left(\log^2 n\right)$ rounds [20]. We note that these results hold in a distributed message passing setting, where processors are located at graph nodes and can communicate only with neighbors. In such a distributed setting, $\Omega\left(\sqrt{\log n/\log\log n}\right)$ time lower bound is known for computing any constant approximation to maximum matching [30].

So far, in the MPC setting, the prior results are due to Lattanzi, Moseley, Suri, and Vassilvitskii [31], Ahn and Guha [1] and Assadi and Khanna [6]. Lattanzi et al. [31] put forth algorithms for several graph problems, such as connected components, minimum spanning tree, and maximum matching problem, that were based on a so-called *filtering technique*. In particular, using this technique, they have obtained an algorithm that can compute a 2-approximation to maximum matching in $O(1/\delta)$ MPC rounds, provided S, the space per machine, is significantly larger than the total number of vertices n, that is $S = \Omega\left(n^{1+\delta}\right)$, for some constant $\delta \in (0,1)$. Later on, Ahn and Guha [1] provided an improved algorithm that computes a $(1+\epsilon)$-approximation in $O(1/(\delta\epsilon))$ rounds, provided $S = \Omega\left(n^{1+\delta}\right)$, for some constant $\delta > 0$. Both these results, however, crucially require that space per machine is significantly superlinear in n, the number of vertices. In fact, if the space S is linear in n, which is a very natural setting for massively parallel graph algorithms, the performance of both these algorithms degrades to $O(\log n)$ parallel rounds, which matches what was known for the PRAM model. Recently, Assadi and Khanna [6] showed how to construct randomized composable coresets of size $\tilde{O}(n)$ that give an $O(1)$-approximation for maximum matching. Their techniques apply to the MPC model only if the space per machine is $\tilde{O}(n\sqrt{n})$.

We also note that the known PRAM maximal independent set and maximal matching algorithms [2, 26, 33] can be used to find a maximal matching (i.e., 2-approximation to maximum matching) in $O(\log n)$ MPC rounds as long as space per machine is at least $n^{\Omega(1)}$ (i.e., $S \geq n^c$ for some constant $c > 0$). We omit further details here, except mentioning that a more or less direct simulation of those algorithms is possible via an $O(1)$-round sorting subroutine [21].

The above results give rise to the following fundamental question: Can the maximum matching be (approximately) solved in $o(\log n)$ parallel rounds in $O(n)$ space per machine? The main result of this paper is an affirmative answer to that question. We show that, for any $S = \Omega(n)$, one can obtain an $O(1)$-approximation to maximum matching using $O\left((\log\log n)^2\right)$ parallel MPC rounds. So, not only do we break the existing $\Omega(\log n)$ barrier, but also provide an almost exponential improvement over the previous work. Our algorithm can also provide a $(2+\epsilon)$, instead of $O(1)$-approximation, at the expense of the number of parallel rounds increasing by a factor of $O(\log(1/\epsilon))$. Finally, our approach can also provide algorithms that have $o(\log n)$ parallel round complexity also in the regime of S being (mildly) sublinear. For instance, we obtain $O\left((\log\log n)^2\right)$ MPC rounds even if space per machine is $S = n/(\log n)^{O(\log\log n)}$. The exact comparison of our bounds with previous results is given in Table 1.

1.1 The Model

In this work, we adopt a version of the model introduced by Karloff, Suri, and Vassilvitskii [29] and refined in later works [3, 8, 21]. We call it *massive parallel computation* (MPC), which is a mutation of the name proposed by Beame et al. [8].

In the MPC model, we have m machines at our disposal and each of them has S words of space. Initially, each machine receives its share of the input. In our case, the input is a collection E of edges and each machine receives approximately $|E|/m$ of them.

The computation proceeds in *rounds*. During the round, each of the machines processes its local data without communicating with other machines. At the end of each round, machines exchange messages. Each message is sent only to a single machine specified by the machine that is sending the message. All messages sent and received by each machine in each round have to fit into the machine's local memory. Hence, their total length is bounded by S.[1] This in particular implies that the total communication of the MPC model is bounded by $m \cdot S$ in each round. The messages are processed by recipients in the next round.

At the end of the computation, machines collectively output the solution. The data output by each machine has to fit in its local memory. Hence again, each machine can output at most S words.

The Range of Values for S and m. If the input is of size N, one usually wants S sublinear in the N, and the total space across all

[1] This for instance allows a machine to send a single word to $S/100$ machines or $S/100$ words to one machine, but not $S/100$ words to $S/100$ machines if $S = \omega(1)$, even if the messages are identical.

Table 1: Comparison of our results for computing approximate maximum size matchings to the previous results for the MPC model.

Source	Approx.	Space	Rounds	Remarks
[31]	2	$n^{1+\Omega(1)}$	$O(1)$	Maximal matching
		$O(n)$	$O(\log n)$	
[1]	$1+\epsilon$	$O\left(n^{1+1/p}\right)$	$O(p/\epsilon)$	$p > 1$
	2	$n^{\Omega(1)}$	$O(\log n)$	Maximal matching Simulate [2, 26, 33]
here	$O(1)$	$O(n)$	$O\left((\log\log n)^2\right)$	$\epsilon \in (0, 1/2)$ $2 \le f(n) = O\left(n^{1/2}\right)$
	$2+\epsilon$		$O\left((\log\log n)^2 \cdot \log(1/\epsilon)\right)$	
	$O(1)$	$O(n)/f(n)$	$O\left((\log\log n)^2 + \log f(n)\right)$	
	$2+\epsilon$		$O\left((\log\log n)^2 + \log f(n)\right) \cdot \log(1/\epsilon)$	

the machines to be at least N—so the input fits onto the machines—and ideally not much larger. Formally, one usually considers $S \in \Theta\left(N^{1-\epsilon}\right)$, for some $\epsilon > 0$.

In this paper, the focus is on graph algorithms. If n is the number of vertices in the graph, the input size can be as large as $\Theta\left(n^2\right)$. Our parallel algorithm requires $\Theta(n)$ space per machine (or even slightly less), which is polynomially less than the size of the input for dense graphs.

Sparse Graphs. Many practical large graphs are believed to have only $O(n)$ edges. One natural example is social networks, in which most participants are likely to have a bounded number of friends. The additional advantage of our approach is that it allows for a small number of processing rounds even if a sparse input graph does not fit onto a single machine. If a small number—say, $f(n)$—of machines is needed even to store the graph, our algorithm still requires only $O\left((\log\log n)^2 + \log f(n)\right)$ rounds for $O(n/f(n))$ space per machine.

Communication vs. Computation Complexity. The main focus of this work is the number of (communication) rounds required to finish computation. Also, even though we do not make an effort to explicitly bound it, it is apparent from the design of our algorithms that every machine performs $O(S \operatorname{polylog} S)$ computation steps locally. This in particular implies that the overall work across all the machines is $O(rN \operatorname{polylog} S)$, where r is the number of rounds and N is the input size (i.e., the number of edges).

The total communication during the computation is $O(rN)$ words. This is at most $O\left(rn^2\right)$ words and it is known that computing a $(1+\epsilon)$-approximate matching in the message passing model with $\Theta(n)$ edges per player may require $\Omega\left(n^2/(1+\epsilon)^2\right)$ bits of communication [24]. Since our value of r is $O\left((\log\log n)^2\right)$ when $\Theta(n)$ edges are assigned to each player, we lose a factor of $\tilde{\Theta}(\log n)$ compared to this lower bound if words (and vertex identifiers) have $\Theta(\log n)$ bits.

1.2 Our Results

In our work, we focus on computing an $O(1)$-approximate maximum matching in the MPC model. We collect our results and compare to the previous work in Table 1. The table presents two interesting regimes for our algorithms. On the one hand, when the space per machine is $S = O(n)$, we obtain an algorithm that requires $O((\log\log n)^2)$ rounds. This is the first known algorithm that, with linear space per machine, breaks the $O(\log n)$ round barrier. On the other hand, in the mildly sublinear regime of space per machine, i.e., when $S = O(n/f(n))$, for some function $f(n)$ that is $n^{o(1)}$, we obtain an algorithm that still requires $o(\log n)$ rounds. This, again, is the first such result in this regime. In particular, we prove the following result.

Theorem 1.1. *There exists an MPC algorithm that constructs an $O(1)$-approximation to maximum matching with constant probability in $O\left((\log\log n)^2 + \max\left(\log \frac{n}{S}, 0\right)\right)$ rounds, where $S = n^{\Omega(1)}$ is the amount of space on each machine.*

As a corollary, we obtain the following result that provides nearly 2-approximate maximum matching.

Corollary 1.2. *For any $\epsilon \in (0, \frac{1}{2})$, there exists an MPC algorithm that constructs a $(2+\epsilon)$-approximation to maximum matching with constant probability in $O\left((\log\log n)^2 + \max\left(\log \frac{n}{S}, 0\right)\right) \cdot \log(1/\epsilon)$ rounds, where $S = n^{\Omega(1)}$ is the amount of space on each machine.*

Assadi et al. [5] observe that one can use a technique of McGregor [34] to extend our algorithm to compute a $(1+\epsilon)$-approximation in $O((\log\log n)^2) \cdot (1/\epsilon)^{O(1/\epsilon)}$ rounds.

It should also be noted that (as pointed out to us by Seth Pettie) any $O(1)$-approximation algorithm for unweighted matchings can be used to obtain a $(2+\epsilon)$-approximation algorithm for weighted matchings (see Section 4 of his paper with Lotker and Patt-Shamir [32] for details). In our setting this implies that Theorem 1.1 yields an algorithm that computes a $(2+\epsilon)$-*approximation* to maximum weight matching in $O((\log\log n)^2 \cdot (1/\epsilon))$ rounds and $O(n \log n)$ space per machine.

1.3 Related Work

We note that there were efforts at modeling MapReduce computation [19] before the work of Karloff et al. Also a recent work [37] investigates the complexity of the MPC model.

In the *filtering* technique, introduced by Lattanzi et al. [31], the input graph is iteratively sparsified until it can be stored on a single machine. For the matching problem, the sparsification is achieved by first obtaining a small sample of edges, then finding

a maximal matching in the sample, and finally removing all the matched vertices. Once a sufficiently small graph is obtained, a maximal matching is computed on a single machine. In the $S = \Theta(n)$ regime, the authors show that their approach reduces the number of edges by a constant factor in each iteration. Despite this guarantee, until the very last step, each iteration may make little progress towards obtaining even an approximate maximal matching, resulting in a $O(\log n)$ round complexity of the algorithm. Similarly, the results of Ahn and Guha [1] require $n^{1+\Omega(1)}$ space per machine to compute a $O(1)$-approximate maximum weight matching in a constant number of rounds and do not imply a similar bound for the case of linear space.

We note that the algorithm of Lattanzi et al. [31] cannot be turned easily into a fast approximation algorithm when space per machine is sublinear. Even with $\Theta(n)$ space, their method is able to remove only a constant fraction of edges from the graph in each iteration, so $\Omega(\log n)$ rounds are needed until only a matching is left. When $S = \Theta(n)$, their algorithm works as follows: sample uniformly at random $\Theta(n)$ edges of the graph, find maximal matching on the sampled set, remove the matched vertices, and repeat. We do not provide a formal proof here, but on the following graph this algorithm requires $\tilde{\Omega}(\log n)$ rounds, even to discover a constant factor approximation. Consider a graph consisting of t separate regular graphs of degree 2^i, for $0 \leq i \leq t-1$, each on 2^t vertices. This graph has $t2^t$ nodes and the algorithm requires $\tilde{\Omega}(t)$ rounds even to find a constant approximate matching. The algorithm chooses edges uniformly at random, and few edges are selected each round from all but the densest remaining subgraphs. Thus, it takes multiple rounds until a matching of significant size is constructed for sparser subgraphs. This example emphasizes the weakness of direct edge sampling and motivates our vertex sampling scheme that we introduce in this paper.

Similarly, Ahn and Guha [1] build on the filtering approach of Lattanzi et al. and design a primal-dual method for computing a $(1+\epsilon)$-approximate weighted maximum matching. They show that each iteration of their distributed algorithm either makes large progress in the dual, or they can construct a large approximate matching. Regardless of their new insights, their approach is inherently edge-sampling based and does not break the $O(\log n)$ round complexity barrier when $S = O(n)$.

Despite the fact that MPC model is rather new, computing matching is an important problem in this model, as the above mentioned two papers demonstrate. This is further witnessed by the fact that the distributed and parallel complexity of maximal matching has been studied for many years already. The best deterministic PRAM maximal matching algorithm, due to Israeli and Shiloach [27], runs in $O\left(\log^3 n\right)$ rounds. Israeli and Itai [26] gave a randomized algorithm for this problem that runs in $O(\log n)$ rounds. Their algorithm works as well in CONGEST, a distributed message-passing model with a processor assigned to each vertex and a limit on the amount of information sent along each edge per round. A more recent paper by Lotker, Patt-Shamir, and Pettie [32] gives a $(1+\epsilon)$-approximation to maximum matching in $O(\log n)$ rounds also in the CONGEST model, for any constant $\epsilon > 0$. On the deterministic front, in the LOCAL model, which is a relaxation of CONGEST that allows for an arbitrary amount of data sent along each edge, a line of research initiated by Hańćkowiak, Karoński, and Panconesi [22, 23] led to an $O\left(\log^3 n\right)$-round algorithm by Fischer and Ghaffari [20].

On the negative side, Kuhn, Moscibroda, and Wattenhofer [30] showed that any distributed algorithm, randomized or deterministic, that performs communication only between neighbors requires $\Omega\left(\sqrt{\log n / \log\log n}\right)$ rounds to compute a constant approximation to maximum matching. This lower bound applies to all distributed algorithms that have been mentioned above. Our algorithm circumvents this lower bound by loosening the only possible assumption there is to be loosened: single-hop communication. In a sense, we assign subgraphs to multiple machines and allow multi-hop communication between nodes in each subgraph.

Finally, the ideas behind the peeling algorithm that is a starting point for this paper can be traced back to the papers of Israeli, Itai, and Shiloach [26, 27], which can be interpreted as matching high-degree vertices first in order to reduce the maximum degree. A sample distributed algorithm given in a work of Parnas and Ron [36] uses this idea to compute an $O(\log n)$ approximation for vertex cover. Their algorithm was extended by Onak and Rubinfeld [35] in order to provide an $O(1)$-approximation for vertex cover and maximum matching in a dynamic version of the problems. This was achieved by randomly matching high-degree vertices to their neighbors in consecutive phases while reducing the maximum degree in the remaining graph. This approach was further developed in the dynamic graph setting by a number of papers [12–15]. Ideas similar to those in the paper of Parnas and Ron [36] were also used to compute polylogarithmic approximation in the streaming model by Kapralov, Khanna, and Sudan [28]. Our version of the peeling algorithm was directly inspired by the work of Onak and Rubinfeld [35] and features important modifications in order to make our analysis go through.

1.4 Future Challenges

We show a parallel matching algorithm in the MPC model by taking an algorithm that can be seen as a distributed algorithm in the so-called LOCAL model. This algorithm requires $\Theta(\log n)$ rounds and can be simulated in $\Theta(\log n)$ MPC rounds relatively easily with $n^{\Omega(1)}$ space per machine. We develop an approximate version of the algorithm that uses much fewer rounds by repeatedly compressing a superconstant number of rounds of the original algorithm to $O(1)$ rounds. It is a great question if this kind of speedup can be obtained for other—either distributed or PRAM—algorithms.

As for the specific problem considered in this paper, an obvious question is whether our round complexity is optimal. We conjecture that there is a better algorithm that requires $O(\log\log n)$ rounds, the square root of our complexity. Unfortunately, a factor of $\log n$ in one of our functions (see the logarithmic factor in α, a parameter defined later in the paper) propagates to the round complexity, where it imposes a penalty of $\log\log n$.

Note also that as opposed to the paper of Onak and Rubinfeld [35], we do not obtain an $O(1)$-approximation to vertex cover. This stems from the fact that we discard so-called reference sets, which can be much bigger than the minimum vertex cover. This is unfortunately necessary in our analysis. Is there a way to fix this shortcoming of our approach?

Finally, we suspect that there is a simpler algorithm for the problem that avoids the intricacies of our approach and proceeds by simply greedily matching high-degree vertices on induced subgraphs without sophisticated sampling in every phase. Unfortunately, we do not know how to analyze this kind of approach.

1.5 Recent Developments

Since an earlier version of this work was shared on arXiv, it has inspired two followup works. First, Assadi [4] applied the round compression idea to the distributed $O(\log n)$-approximation algorithm for vertex cover of Parnas and Ron [36]. Using techniques from his recent work with Khanna [6], he gave a simple MPC algorithm that in $O(\log \log n)$ rounds and $n/\text{polylog}(n)$ space per machine computes an $O(\log n)$-approximation to minimum vertex cover.

Second, a new paper by Assadi et al. [5] addresses, among other things, several open questions from this paper. They give an MPC algorithm that computes $O(1)$-approximation to both vertex cover and maximum matching in $O(\log \log n)$ rounds and $\tilde{O}(n)$ space per machine (though the space is strictly superlinear). Their result builds on techniques developed originally for dynamic matching algorithms [10, 11] and composable coresets [6]. It is worth to note that their construction critically relies on the vertex sampling approach (i.e., random assignment of vertices to machines) introduced in our work.

2 OVERVIEW

In this section we present the main ideas and techniques behind our result. Our paper contains two main technical contributions.

First, our algorithm *randomly partitions vertices* across the machines, and on each machine considers only the corresponding induced graph. We prove that it suffices to consider these induced subgraphs to obtain an approximate maximum matching. Note that this approach greatly deviates from previous works, that used edge based partitioning.

Second, we introduce a *round compression* technique. Namely, we start with an algorithm that executes $O(\log n)$ phases and can be naturally implemented in $O(\log n)$ MPC rounds and then demonstrate how to emulate this algorithm using only $o(\log n)$ MPC rounds. The underlying idea is quite simple: each machine independently runs multiple phases of the initial algorithm. This approach, however, has obvious challenges since the machines cannot communicate in a single round of the MPC algorithm. The rest of the section is devoted to describing our approach and illustrating how to overcome these challenges.

2.1 Vertex Based Sampling

The algorithms for computing maximal matching in PRAM and their simulations in the MPC model [2, 26, 27, 33] are designed to, roughly speaking, either halve the number of the edges or halve the maximum degree in each round. Therefore, in the worst case those algorithms inherently require $\Omega(\log n)$ rounds to compute a maximal matching.

On the other hand, all the algorithm for the maximal matching problem in the MPC model prior to ours ([1, 6, 31]) process the input graph by discarding edges, and eventually aggregate the remaining edges on a single machine to decide which of them are part of the final matching. It is not known how to design approaches similar to [1, 6, 31] while avoiding a step in which the maximal matching computation is performed on a single machine. This seems to be a barrier for improving upon $O(\log n)$ rounds, if the space available on each machine is $O(n)$.

The starting point of our new approach is alleviating this issue by resorting to a more careful vertex based sampling. Specifically, at each round, we randomly partition the vertex set into vertex sets $V_1, \ldots, V_m$ and consider induced graphs on those subsets independently. Such sampling scheme has the following handy property: the union of matchings obtained across the machines is still a matching. Furthermore, we show that for the appropriate setting of parameters this sampling scheme allows us to handle vertices of a wide range of degrees in a single round, unlike handling only high-degree vertices (that is, vertices with degree within a constant factor of the maximum degree) as guaranteed by [26, 27].

2.2 Global algorithm

To design an algorithm executed on machines locally, we start from a sequential peeling algorithm `GlobalAlg` (see Algorithm 1), which is a modified version of an algorithm used by Onak and Rubinfeld [35]. The algorithm had to be significantly adjusted in order to make our later analysis of a parallel version possible.

The execution of `GlobalAlg` is divided into $\Theta(\log n)$ *phases*. In each phase, the algorithm first computes a set H of high-degree vertices. Then it selects a set F of vertices, which we call *friends*. Next the algorithm selects a matching $\widetilde{M}$ between H and F, using a simple randomized strategy. F is carefully constructed so that both F and $\widetilde{M}$ are likely to be of order $\Theta(|H|)$. Finally, the algorithm removes all vertices in $H \cup F$, hence reducing the maximum vertex degree in the graph by a constant factor, and proceeds to the next phase. The central property of `GlobalAlg` is that it returns an $O(1)$ approximation to maximum matching with constant probability (Corollary 3.4). A detailed discussion of `GlobalAlg` is given in Section 3.

Algorithm 1: `GlobalAlg`$(G, \widetilde{\Delta})$
Global matching algorithm

Input: Graph $G = (V, E)$ of maximum degree at most $\widetilde{\Delta}$
Output: A matching in G

$\Delta \leftarrow \widetilde{\Delta}$, $M \leftarrow \emptyset$, $V' \leftarrow V$
while $\Delta \geq 1$ **do**
  `/* Invariant: the maximum degree in` $G[V']$ `is at most` Δ `*/`
  Let $H \subset V'$ be a set of vertices of degree at least $\Delta/2$ in $G[V']$. We call vertices in H *heavy*.
  Create a set F of *friends* by selecting each vertex $v \in V'$ independently with probability $|N(v) \cap H|/4\Delta$.
  Compute a matching $\widetilde{M}$ in $G[H \cup F]$ using `MatchHeavy`(H, F) and add it to M.
  $V' \leftarrow V' \setminus (H \cup F)$, $\Delta \leftarrow \Delta/2$
return M

Algorithm 2: `MatchHeavy`(H, F)
Computing a matching in $G[H \cup F]$

Input: set H of heavy vertices and set F of friends
Output: a matching in $G[H \cup F]$

1 For every vertex $v \in F$ pick uniformly at random a heavy neighbor $v_\star$ in $N(v) \cap H$.
2 Independently at random color each vertex in $H \cup F$ either red or blue.
3 Select the following subset of edges:
$E_\star \leftarrow \{(v, v_\star) : v \in F \wedge v \text{ is red} \wedge v_\star \in H \wedge v_\star \text{ is blue}\}$.
4 For every blue vertex w incident to an edge in $E_\star$, select one such edge and add it to $\widetilde{M}$.
5 **return** $\widetilde{M}$

2.3 Parallel Emulation of the Global Algorithm (Section 4)

The following two ways could be used to execute `GlobalAlg` in the MPC model: (1) place the whole graph on one machine, and trivially execute all the phases of `GlobalAlg` in a single round; or (2) simulate one phase of `GlobalAlg` in one MPC round while using $O(n)$ space per machine, by distributing vertices randomly onto machines. However, each of these approaches has severe drawbacks. The first approach requires $\Theta(|E|)$ space per machine, which is likely prohibitive for large graphs. On the other hand, while the second approach uses $O(n)$ space, it requires $\Theta(\log n)$ rounds of MPC computation. We achieve the best of both worlds by showing how to emulate the behavior of *multiple phases* of `GlobalAlg` in a *single MPC round* with each machine using $O(n)$ space, thus obtaining an MPC algorithm requiring $o(\log n)$ rounds. More specifically, we show that it is possible to emulate the behavior of `GlobalAlg` in $O\left((\log\log n)^2\right)$ rounds with each machine using $O(n)$ (or even only $n/(\log n)^{O(\log\log n)}$) space.

Before we provide more details about our parallel multi-phase emulation of `GlobalAlg`, let us mention the main obstacle such an emulation encounters. At the beginning of every phase, `GlobalAlg` has access to the full graph. Therefore, it can easily compute the set of heavy vertices H. On the other hand, machines in our MPC algorithm use $O(n)$ space and thus have access only to a small subgraph of the input graph (when $|E| \gg n$). In the first phase this is not a big issue, as, thanks to randomness, each machine can estimate the degrees of high-degree vertices. However, the degrees of vertices can significantly change from phase to phase. Therefore, after each phase it is not clear how to select high-degree vertices in the next phase without inspecting the entire graph again. Hence, one of the main challenges in designing a multi-phase emulation of `GlobalAlg` is to ensure that machines at the beginning of every phase can estimate *global* degrees of vertices well enough to identify the set of heavy vertices, while each machine still having access only to its local subgraph. This property is achieved using a few modifications to the algorithm.

2.3.1 Preserving Randomness. Our algorithm partitions the vertex set into m disjoint subsets V_i by assigning each vertex independently and uniformly at random. Then the graph induced by each subset V_i is processed on a separate machine. Each machine finds a set of heavy vertices, H_i, by estimating the global degree of each vertex of V_i. It is not hard to argue (using a standard concentration bound) that there is enough randomness in the initial partition so that local degrees in each induced subgraph roughly correspond to the global degrees. Hence, after the described partitioning, sets H and $\bigcup_{i\in[m]} H_i$ have very similar properties. This observation crucially relies on the fact that initially the vertices are distributed *independently* and *uniformly* at random.

However, if the second phase of `GlobalAlg` is executed without randomly reassigning vertices to sets after the first phase, the remaining vertices are no longer distributed independently and uniformly at random. In other words, after inspecting the neighborhood of every vertex locally and making a decision based on it, the randomness of the initial random partition may significantly decrease.

Let us now make the following thought experiment. Imagine for a moment that there is an algorithm that emulates multiples phases of `GlobalAlg` in parallel and in every phase inspects **only** the vertices that end-up being matched. Then, from the point of view of the algorithm, the vertices that are not matched so far are still distributed independently and uniformly at random across the machines. Or, saying in a different way, if randomness of some vertices is not inspected while emulating a phase, then at the beginning of the next phase those vertices still have the same distribution as in the beginning of that MPC round. But, how does an algorithm learn about vertices that should be matched by inspecting no other vertex? How does the algorithm learn even only about high-degree vertices without looking at their neighborhood?
In the sequel we show how to design an algorithm that looks only "slightly" at the vertices that do not end-up being matched. As we prove, that is sufficient to design a multi-phase emulation of `GlobalAlg`.

We now discuss in more detail how to preserve two crucial properties of our vertex assignments throughout the execution of multiple phases: independent and nearly-uniform distribution.

2.3.2 Independence (Lemma 4.2). As noted above, it is not clear how to compute vertex degrees without inspecting their local neighborhood. A key, and at first sight counter-intuitive, step in our approach is to *estimate* even *local degrees* of vertices (in contrast to computing them exactly). To obtain the estimates, it suffices to examine only small neighborhoods of vertices and in turn preserve the independent distribution of the intact ones. More precisely, we sample a small set of vertices on each machine, called *reference sets*, and use the set to estimate the local degrees of all vertices assigned to this machine. Furthermore, we show that with a proper adjustments of `GlobalAlg` these estimates are sufficient for capturing high-degree vertices.

Very crucially, all the vertices that are used in computing a matching in one emulated phase (including the reference sets) are discarded at the end of the phase, even if they do not participate in the obtained matching. In this way we disregard the vertices which position is fixed and, intuitively, secure an independent distribution of the vertices across the machines in the next phase.

We also note, without going into details, that obtaining full independence required modifying how the set of friends is selected, compared to the original approach of Onak and Rubinfeld [35]. In their approach, each heavy vertex selected one friend at random.

However, as before, in order to select exactly one friend would require examining neighborhood of heavy vertices. This, however, introduces dependencies between vertices that have not been selected. So instead, in our GlobalAlg, every vertex selects itself as a friend independently and proportionally to the number of high-degree vertices (found using the reference set), which again secures an independent distribution of the remaining vertices. The final properties of the obtained sets in either approach are very similar.

2.3.3 Uniformity (Lemma 4.3). A very convenient property in the task of emulating multiple phases of GlobalAlg is a uniform distribution of vertices across all the machines at every phase – for such a distribution, we know the expected number of neighbors of each desired type assigned to the same machine. Obtaining perfect uniformity seems difficult—if not impossible in our setting—and we therefore settle for *near* uniformity of vertex assignments. The probability of the assignment of each vertex to each machine is allowed to differ slightly from that in the uniform distribution. Initially, the distribution of each vertex is uniform and with every phase it can deviate more and more from the uniform distribution. We bound the rate of the decay with high probability and execute multiple rounds as long as the deviation from the uniform distribution is negligible. More precisely, in the execution of the entire parallel algorithm, the sufficiently uniform distribution is on average kept over $\Omega\left(\frac{\log n}{(\log\log n)^2}\right)$ phases of the emulation of GlobalAlg.

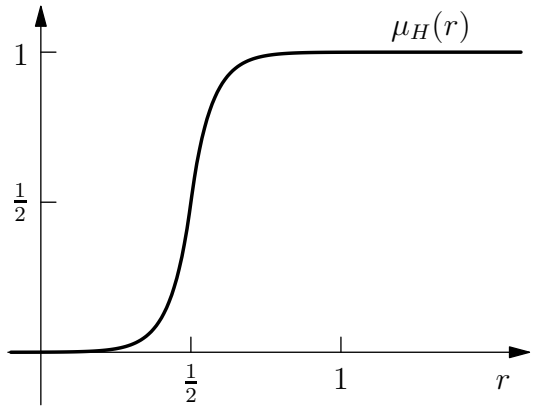

Figure 1: An idealized version of $\mu_H : \mathbb{R} \to [0, 1]$, in which n was fixed to a small constant and the multiplicative constant inside the exponentiation operator was lowered.

In order to achieve the near uniformity, we modify the procedure for selecting H, the set of high-degree vertices. Instead of a hard threshold on the degrees of vertices that are included in H as in the sequential algorithm, we randomize the selection by using a carefully crafted threshold function μ_H. This function specifies the probability with which a vertex is included in H. It takes as input the ratio of the vertex's degree to the current maximum degree (or, more precisely, the current upper bound on the maximum degree) and it smoothly transitions from 0 to 1 in the neighborhood of the original hard threshold (see Figure 1). The main intuition behind the introduction of this function is that we want to ensure that a vertex is *not* selected for H with almost the same probability, independently of the machine on which it resides. Using a hard threshold instead of μ_H could result in the following deficiency. Consider a vertex v that has slightly too few neighbors to qualify as a heavy vertex. Still, it could happen, with a non-negligible probability, that the reference set of some machine contains so many neighbors of v that v would be considered heavy on this machine. However, if v is not included in the set of heavy vertices on that machine, it becomes clear after even a single phase that the vertex is not on the given machine, i.e. the vertex is on the given machine with probability zero. At this point the distribution is clearly no longer uniform.

Function μ_H has further useful properties that we extensively exploit in our analysis. We just note that in order to ensure near uniformity with high probability, we also have to ensure that each vertex is selected for F, the set of friends, with roughly the same probability on each machine.

2.4 Organization of This Extended Abstract

We start by analyzing GlobalAlg in Section 3. Section 4 describes how to emulate a single phase of GlobalAlg in the MPC model. Section 5 gives and analyzes our parallel algorithm by putting together components developed in the previous sections. The resulting parallel algorithm can be implemented in the MPC model in a fairly straightforward way by using the result of [21]. The details of the implementation and missing proofs are given in the full version of this paper (see also https://arxiv.org/abs/1707.03478).

3 GLOBAL ALGORITHM

3.1 Overview

The starting point of our result is a peeling algorithm GlobalAlg that takes as input a graph G, and removes from it vertices of lower and lower degree until no edge is left. See page 5 for its pseudocode. We use the term *phase* to refer to an iteration of the main loop in Lines 2–6.

Each phase is associated with a threshold Δ. Initially, Δ equals $\widetilde{\Delta}$, the upper bound on the maximum vertex degree. In every phase, Δ is divided by two until it becomes less than one and the algorithm stops. Since during the execution of the algorithm we maintain the invariant that the maximum degree in the graph is at most Δ, the graph has no edge left when the algorithm terminates.

In each phase the algorithm matches, in expectation, a constant fraction of the vertices it removes. We use this fact to prove that, across all the phases, the algorithm computes a constant-factor approximate matching. In the rest of this section, we analyze this algorithm.

3.2 Analysis

We start our analysis of the algorithm by showing that the execution of MatchHeavy in each phase of GlobalAlg finds a relatively large matching in expectation.

Lemma 3.1. *Consider one phase of* GlobalAlg. *Let H be the set of heavy vertices.* MatchHeavy *finds a matching $\widetilde{M}$ such that $\mathbb{E}\left[\left|\widetilde{M}\right|\right] \geq \frac{1}{40}|H|$.*

Proof. Observe that the set $E_\star$ is a collection of vertex-disjoint stars: each edge connects a red vertex with a blue vertex and the red vertices have degree 1. Thus, a subset of $E_\star$ forms a valid matching as long as no blue vertex is incident to two matched edges. Note that this is guaranteed by how edges are added to $\widetilde{M}$ in Line 4.

The size of the computed matching is the number of blue vertices in H that have at least one incident edge in $E_\star$. Let us now lower bound the number of such vertices. Consider an arbitrary $u \in H$. It has the desired properties exactly when the following three independent events happen: some v is selected in F and v selects u in Line 1; u is colored blue; and v is colored red. The joint probability of the two latter events is exactly $\frac{1}{4}$. The probability that u is *not* selected by some its neighbor v (either because v is not selected in F, or v is selected in F but v does not select u in Line 1) is

$$\left(1-\frac{1}{4\Delta}\right)^{|N(u)\cap V'|} \le \left(1-\frac{1}{4\Delta}\right)^{\Delta/2} \le \exp\left(-\frac{1}{8}\right) \le \frac{9}{10}.$$

This implies that u is selected by a neighbor $v \in F$ with probability at least $\frac{1}{10}$. Therefore, with probability at least $\frac{1}{10}\cdot\frac{1}{4} = \frac{1}{40}$, u is blue and incident to an edge in $E_\star$. Hence, $\mathbb{E}\left[\left|\widetilde{M}\right|\right] \ge \frac{1}{40}|H|$. □

Next we show an upper bound on the expected size of F, the set of friends.

Lemma 3.2. *Let H be the set of heavy vertices selected in a phase of* `GlobalAlg`*. The following bound holds on the expected size of F, the set of friends, created in the same phase: $\mathbb{E}[|F|] \le \frac{1}{4}|H|$.*

Proof. At the beginning of a phase, every vertex $u \in V'$—including those in H—has its degree, $|N(u)\cap V'|$, bounded by Δ. Reversing the order of the summation and applying this fact, we get:

$$\mathbb{E}[|F|] = \sum_{v\in V'} \frac{|N(v)\cap H|}{4\Delta} = \sum_{u\in H} \frac{|N(u)\cap V'|}{4\Delta} \le \frac{|H|\cdot\Delta}{4\Delta} = \frac{|H|}{4}.$$

□

We combine the last two bounds to lower bound the expected size of the matching computed by `GlobalAlg`.

Lemma 3.3. *Consider an input graph G with an upper bound $\widetilde{\Delta}$ on the maximum vertex degree. Assume that* `GlobalAlg`*$(G,\widetilde{\Delta})$ executes $T \stackrel{\text{def}}{=} \lfloor\log\widetilde{\Delta}\rfloor + 1$ phases. Let H_i, F_i, and $\widetilde{M}_i$ be the sets H, F, and $\widetilde{M}$ constructed in phase i for $i \in [T]$. The following relationship holds on the expected sizes of these sets:*

$$\sum_{i=1}^{T} \mathbb{E}\left[\left|\widetilde{M}_i\right|\right] \ge \frac{1}{50}\sum_{i=1}^{T}\mathbb{E}[|H_i|+|F_i|]$$

Proof. For each phase $i \in [T]$, by applying the expectation over all possible settings of the set H_i, we learn from Lemmas 3.1 and 3.2 that

$$\mathbb{E}\left[\left|\widetilde{M}_i\right|\right] \ge \frac{1}{40}\mathbb{E}[|H_i|] \qquad \text{and} \qquad \mathbb{E}[|F_i|] \le \frac{1}{4}\mathbb{E}[|H_i|].$$

It follows that

$$\frac{1}{50}\mathbb{E}[|H_i|+|F_i|] \le \frac{1}{50}\mathbb{E}[|H_i|]+\frac{1}{200}\mathbb{E}[|H_i|] = \frac{1}{40}\mathbb{E}[|H_i|] \le \mathbb{E}\left[\left|\widetilde{M}_i\right|\right],$$

and the statement of the lemma follows by summing over all phases. □

We do not use this fact directly in our paper, but note that the last lemma can be used to show that `GlobalAlg` can be used to find a large matching.

Corollary 3.4. `GlobalAlg` *computes a constant factor approximation to the maximum matching with $\Omega(1)$ probability.*

Proof. First, note that `GlobalAlg` finds a correct matching, i.e., no two different edges in M share an endpoint. This is implied by the fact that M is extended in every phase by a matching on a disjoint set of vertices.

Let T and sets H_i, F_i, and $\widetilde{M}_i$ for $i \in [T]$ be defined as in the statement of Lemma 3.3. Let M_{OPT} be a maximum matching in the graph. Observe that at the end of the algorithm execution, the remaining graph is empty. This implies that the size of the maximum matching can be bounded by the total number of removed vertices, because each removed vertex decreases the maximum matching size by at most one:

$$\sum_{i=1}^{T} |H_i| + |F_i| \ge |M_{\text{OPT}}|.$$

Hence, using Lemma 3.3,

$$\mathbb{E}[|M|] = \sum_{i=1}^{T}\mathbb{E}\left[\left|\widetilde{M}_i\right|\right] \ge \frac{1}{50}\sum_{i=1}^{T}\mathbb{E}[|H_i|+|F_i|] \ge \frac{1}{50}|M_{\text{OPT}}|.$$

Since $|M| \le |M_{\text{OPT}}|$, $|M| \ge \frac{1}{100}|M_{\text{OPT}}|$ with probability at least $\frac{1}{100}$. Otherwise, $\mathbb{E}[|M|]$ would be strictly less than $\frac{1}{100}\cdot|M_{\text{OPT}}| + 1\cdot\frac{1}{100}|M_{\text{OPT}}| = \frac{1}{50}|M_{\text{OPT}}|$, which is not possible. □

4 EMULATION OF A PHASE IN A RANDOMLY PARTITIONED GRAPH

In this section, we introduce a modified version of a single phase (one iteration of the main loop) of `GlobalAlg`. Our modifications later allow for implementing the algorithm in the MPC model. The pseudocode of the new procedure, `EmulatePhase`, is presented as Algorithm 3. We partition the vertices of the current graph into m sets V_i, $1 \le i \le m$. Each vertex is assigned independently and *almost* uniformly at random to one of the sets. For each set V_i, we run a subroutine `LocalPhase` (presented as Algorithm 4). This subroutine runs a carefully crafted approximate version of one phase of `GlobalAlg` with an appropriately rescaled threshold Δ. More precisely, the threshold passed to the subroutine is scaled down by a factor of m, which corresponds to how approximately vertex degrees decrease in subgraphs induced by each of the sets. The main intuition behind this modification is that we hope to break the problem up into smaller subproblems on disjoint induced subgraphs, and obtain similar *global* properties by solving the problem approximately on each smaller part. Later, in Section 5, we design an algorithm that assigns the subproblems to different machines and solves them in parallel.

We now discuss `LocalPhase` (i.e., Algorithm 4) in more detail. Table 2 introduces two parameters, α and μ_R, and two functions, μ_H and μ_F, which are used in `LocalPhase`. Note first that α is a parameter used in the definition of μ_H but it is not used in the pseudocode of `LocalPhase` (or `EmulatePhase`) for anything else. It is, however, a convenient abbreviation in the analysis and the later

Algorithm 3: `EmulatePhase`$(\Delta, G_\star, m, \mathcal{D})$
Emulation of a single phase in a randomly partitioned graph

Input:
- threshold Δ
- induced subgraph $G_\star = (V_\star, E_\star)$ of maximum degree $\frac{3}{2}\Delta$
- number m of subgraphs
- ϵ-near uniform and independent distribution $\mathcal{D}$ on assignments of $V_\star$ to $[m]$

Output: Remaining vertices and a matching

1 Pick a random assignment $\Phi : V_\star \to [m]$ from $\mathcal{D}$
2 **for** $i \in [m]$ **do**
3 $\quad V_i \leftarrow \{v \in V_\star : \Phi(v) = i\}$
4 $\quad (V_i', M_i) \leftarrow$ `LocalPhase`$(i, G_\star[V_i], \Delta/m)$ `/* Algorithm 4 */`
5 **return** $\left(\bigcup_{i=1}^m V_i', \bigcup_{i=1}^m M_i\right)$

Table 2: Global parameters $\alpha \in (1, \infty)$ and $\mu_R \in (0, 1)$ and functions $\mu_H : \mathbb{R} \to [0, 1]$ and $\mu_F : \mathbb{R} \to [0, 1]$ used in the parallel algorithm. α, μ_R, and μ_H depend on n, the total number of vertices in the graph.

A multiplicative constant used in the exponent of μ_H:

$$\alpha \stackrel{\text{def}}{=} 96 \ln n.$$

The probability of the selection for a reference set:

$$\mu_R \stackrel{\text{def}}{=} \left(10^6 \cdot \log n\right)^{-1}.$$

The probability of the selection for a heavy set (used with r equal to the ratio of the estimated degree to the current threshold):

$$\mu_H(r) \stackrel{\text{def}}{=} \begin{cases} \frac{1}{2}\exp\left(\frac{\alpha}{2}(r - 1/2)\right) & \text{if } r \le 1/2, \\ 1 - \frac{1}{2}\exp\left(-\frac{\alpha}{2}(r - 1/2)\right) & \text{if } r > 1/2. \end{cases}$$

The probability of the selection for the set of friends (used with r equal to the ratio of the number of heavy neighbors to the current threshold):

$$\mu_F(r) \stackrel{\text{def}}{=} \begin{cases} \max\{r/4, 0\} & \text{if } r \le 4, \\ 1 & \text{if } r > 4. \end{cases}$$

parallel algorithm. The other three mathematical objects specify probabilities with which vertices are included in sets that are created in an execution of `LocalPhase`.

Apart from creating its own versions of H, the set of heavy vertices, and F, the set of friends, `LocalPhase` constructs also a set R_i, which we refer to as a *reference set*. In Line 1, the algorithm puts each vertex in R_i independently and with the same probability μ_R. The reference set is used to estimate the degrees of other vertices in the same induced subgraph in Line 2. For each vertex v_i, its estimate $\widehat{d}_v$ is defined as the number of v's neighbors in R_i multiplied by μ_R^{-1} to compensate for sampling. Next, in Line 3, the algorithm uses the estimates to create H_i, the set of heavy vertices. Recall that `GlobalAlg` uses a sharp threshold for selecting heavy vertices: all vertices of degree at least $\Delta/2$ are placed in H_i. `LocalPhase` works differently. It divides the degree estimate by the current threshold $\Delta_\star$ and uses function μ_H to decide with what probability

Algorithm 4: `LocalPhase`$(i, G_i, \Delta_\star)$
Emulation of a single phase on an induced subgraph

Input:
- induced subgraph number i (useful only for the analysis)
- induced subgraph $G_i = (V_i, E_i)$
- threshold $\Delta_\star \in \mathbb{R}_+$

Output: Remaining vertices and a matching on V_i

1 Create a *reference set* R_i by independently selecting each vertex in V_i with probability μ_R.
2 For each $v \in V_i$, $\widehat{d}_v \leftarrow |N(v) \cap R_i|/\mu_R$.
3 Create a set H_i of *heavy vertices* by independently selecting each $v \in V_i$ with probability $\mu_H\left(\widehat{d}_v/\Delta_\star\right)$.
4 Create a set F_i of *friends* by independently selecting each vertex in $v \in V_i$ with probability $\mu_F\left(|N(v) \cap H_i|/\Delta_\star\right)$.
5 Compute a maximal matching M_i in $G[H_i \cup F_i]$.
6 **return** $(V_i \setminus (R_i \cup H_i \cup F_i), M_i)$

the corresponding vertex is included in H_i. A sketch of the function can be seen in Figure 1. The function transitions from almost 0 to almost 1 in the neighborhood of $\frac{1}{2}$ at a limited pace. As a result vertices of degrees smaller than, say, $\frac{1}{4}\Delta$ are very unlikely to be included in H_i and vertices of degree greater than $\frac{3}{4}\Delta$ are very likely to be included in H_i. `GlobalAlg` can be seen as an algorithm that instead of μ_H, uses a step function that equals 0 for arguments less than $\frac{1}{2}$ and abruptly jumps to 1 for larger arguments. Observe that without μ_H, the vertices whose degrees barely qualify them as heavy could behave very differently depending on which set they were assigned to. We use μ_H to guarantee a smooth behavior in such cases. That is one of the key ingredients that we need for making sure that a set of vertices that remains on one machine after a phase has almost the same statistical properties as a set of vertices obtained by new random partitioning.

Finally, in Line 4, `LocalPhase` creates a set of friends. This step is almost identical to what happens in the global algorithm. The only difference is that this time we have no upper bound on the number of heavy neighbors of a vertex. As a result that number divided by $4\Delta_\star$ can be greater than 1, in which case we have to replace it with 1 in order to obtain a proper probability. This is taken care of by function μ_F. Once H_i and F_i have been created, the algorithm finds a maximal matching M_i in the subgraph induced by the union of these two sets. The algorithm discards from the further consideration not only H_i and F_i, but also R_i. This eliminates dependencies in the possible distribution of assignments of vertices that have not been removed yet if we condition this distribution on the configuration of sets that have been removed. Intuitively, the probability of a vertex's inclusion in any of these sets depends only on R_i and H_i but not on any other vertices. Hence, once we fix the sets of removed vertices, the assignment of the remaining vertices to subgraphs is fully independent.[2] The output of `LocalPhase` is a subset of V_i to be considered in later phases and a matching

[2] By way of comparison, consider observing an experiment in which we toss the same coin twice. The bias of the coin is not fixed but comes from a random distribution. If we do not know the bias, the outcomes of the coin tosses are not independent. However, if we do know the bias, the outcomes are independent, even though they have the same bias.

M_i, which is used to expand the matching that we construct for the entire input graph. We now introduce additional concepts and notation. They are useful for describing and analyzing properties of the algorithm. A configuration describes sets R_i, H_i, and F_i, for $1 \le i \le m$, constructed in an execution of `EmulatePhase`. We use it for conditioning a distribution of vertex assignments as described in the previous paragraph. We also formally define two important properties of distributions of vertex assignments: independence and near uniformity.

Configurations. Let m and $V_\star$ be the parameters to `EmulatePhase`: the number of subgraphs and the set of vertices in the graph to be partitioned, respectively. We say that

$$C = \left(\{R_i\}_{i\in[m]}, \{H_i\}_{i\in[m]}, \{F_i\}_{i\in[m]}\right)$$

is an *m-configuration* if it represents a configuration of sets R_i, H_i, and F_i created by `EmulatePhase` in the simulation of a phase. Recall that for any $i \in [m]$, R_i, H_i, and F_i are the sets created (and removed) by the execution of `LocalPhase` for V_i, the i-th subset of vertices.

We say that a vertex v is *fixed* by C if it belongs to one of the sets in the configuration, i.e.,

$$v \in \bigcup_{i\in[m]} (R_i \cup H_i \cup F_i)\,.$$

Conditional Distribution. Let $\mathcal{D}$ be a distribution on assignments $\varphi : V_\star \to [m]$. Suppose that we execute `EmulatePhase` for $\mathcal{D}$ and let C be a *non-zero probability* m-configuration—composed of sets R_i, H_i, and F_i for $i \in [m]$—that can be created in this setting. Let $V'_\star$ be the set of vertices in $V_\star$ that are *not* fixed by C. We write $\mathcal{D}[C]$ to denote the conditional distribution of possible assignments of vertices in $V'_\star$ to $[m]$, given that for all $i \in [m]$, R_i, H_i, and F_i in C were the sets constructed by `LocalPhase` for the i-th induced subgraph.

Near Uniformity and Independence. Let $\mathcal{D}$ be a distribution on assignments $\varphi : \widetilde{V} \to [m]$ for some set $\widetilde{V}$ and m. For each vertex $v \in \widetilde{V}$, let $p_v : [m] \to [0,1]$ be the probability mass function of the marginal distribution of v's assignment. For any $\epsilon \ge 0$, we say that $\mathcal{D}$ is *ϵ-near uniform* if for every vertex v and every $i \in [m]$, $p_v(i) \in [(1-\epsilon)/m, (1+\epsilon)/m]$. We say that $\mathcal{D}$ is an *independent* distribution if the probability of every assignment φ in $\mathcal{D}$ equals exactly $\prod_{v\in V'} p_v(\varphi(v))$.

4.1 Overview of the Main Lemmas

In this section we state our main lemmas. Their proofs are deferred to the full version of the paper.

We start by showing that `EmulatePhase` computes a large matching as follows. Each vertex belonging to H_i or F_i that `EmulatePhase` removes in the calls to `LocalPhase` can decrease the maximum matching size in the graph induced by the remaining vertices by one. We show that the matching that `EmulatePhase` constructs in the process captures on average at least a constant fraction of that loss. We also show that the effect of removing R_i is negligible. More precisely, we prove the following.

Lemma 4.1. *Let Δ, $G_\star = (V_\star, E_\star)$, m, and $\mathcal{D}$ be parameters for* `EmulatePhase` *such that*

- *$\mathcal{D}$ is an independent and ϵ-near uniform distribution on assignments of vertices $V_\star$ to $[m]$ for $\epsilon \in [0, 1/200]$,*
- *$\frac{\Delta}{m} \ge 4000\mu_R^{-2}\ln^2 n$,*
- *the maximum degree of a vertex in $G_\star$ is at most $\frac{3}{2}\Delta$.*

For each $i \in [m]$, let H_i, F_i, and M_i be the sets constructed by `LocalPhase` *for the i-th induced subgraph. Then, the following relationship holds for their expected sizes:*

$$\sum_{i\in[m]} \mathbb{E}\left[|H_i \cup F_i|\right] \le n^{-9} + 1200 \sum_{i\in[m]} \mathbb{E}\left[|M_i|\right].$$

Note that Lemma 4.1 requires that the vertices are distributed independently and near uniformly in the m sets. This is trivially the case right after the vertices are partitioned independently at random. However, in the final algorithm, after we partition the vertices, we run *multiple* phases on each machine. In the rest of this section we show that running a single phase *preserves* independence of vertex distribution and only slightly disturbs the uniformity (Lemma 4.2 and Lemma 4.3). As we have mentioned before, independence stems from the fact that we use reference sets to estimate vertex degrees. We discard them at the end and condition on them, which leads to the independence of the distribution of vertices that are not removed.

Lemma 4.2. *Let $\mathcal{D}$ be an independent distribution of assignments of vertices in $V_\star$ to $[m]$. Let C be a non-zero probability m-configuration that can be constructed by* `EmulatePhase` *for $\mathcal{D}$. Let $V'_\star$ be the set of vertices of $V_\star$ that are not fixed by C. Then $\mathcal{D}[C]$ is an independent distribution of vertices in $V'_\star$ on $[m]$.*

Independence of the vertex assignment is a very handy feature that allows us to use Chernoff-like concentration inequalities in the analysis of multiple phase emulation. However, although the vertex assignment of non-removed vertices remains independent across machines from phase to phase, as stated by Lemma 4.2, their distribution is not necessarily uniform. Fortunately, we can show it is near uniform.

The proof of near uniformity is the most involved proof in this paper. In a nutshell, the proof is structured as follows. We pick an arbitrary vertex v that has not been removed and show that with high probability it has the same number of neighbors in all sets R_i. The same property holds for v's neighbors in all sets H_i. We use this to show that the probability of a fixed configuration of sets removed in a single phase is roughly the same for all assignments of v to subgraphs. In other words, if v was distributed nearly uniformly before the execution of `EmulatePhase`, it is distributed only slightly less uniformly after the execution.

Lemma 4.3. *Let Δ, $G_\star = (V_\star, E_\star)$, m, and $\mathcal{D}$ be parameters for* `EmulatePhase` *such that*

- *$\mathcal{D}$ is an independent and ϵ-near uniform distribution on assignments of vertices $V_\star$ to $[m]$ for $\epsilon \in [0, (200\ln n)^{-1}]$,*
- *$\frac{\Delta}{m} \ge 4000\mu_R^{-2}\ln^2 n$.*

Let C be an m-configuration constructed by `EmulatePhase`*. With probability at least $1 - n^{-4}$ both the following properties hold:*

- *The maximum degree in the graph induced by the vertices not fixed in C is bounded by $\frac{3}{4}\Delta$.*
- *$\mathcal{D}[C]$ is $60\alpha\left(\left(\frac{\Delta}{m}\right)^{-1/4} + \epsilon\right)$-near uniform.*

5 PARALLEL ALGORITHM

In this section, we introduce our main parallel algorithm. It builds on the ideas introduced in `EmulatePhase`. `EmulatePhase` randomly partitions the graph into m induced subgraphs and runs on each of them `LocalPhase`, which resembles a phase of `GlobalAlg`. As we have seen, the algorithm performs well even if vertices are assigned to subgraphs not exactly uniformly so long as the assignment is fully independent. Additionally, with high probability, if we condition on the configuration of sets R_i, H_i, and F_i that were removed, the distribution of assignments of the remaining vertices is still nearly uniform and also independent.

These properties allow for the main idea behind the final parallel algorithm. We partition vertices randomly into m induced subgraphs and then run `LocalPhase` multiple times on each of them with no repartitioning in the meantime. In each iteration, for a given subgraph, we halve the local threshold $\Delta_\star$. This corresponds to multiple phases of the original global algorithm. As long as we can show that this approach leads to finding a large matching, the obvious gain is that *multiple* phases of the original algorithm translate to $O(1)$ parallel rounds. This approach enables our main result: the parallel round complexity reduction from $O(\log n)$ to $O((\log\log n)^2)$.

Algorithm 5: `ParallelAlg`(G, S)
The final parallel matching algorithm

Input:
- graph $G = (V, E)$ on n vertices
- parameter $S \in \mathbb{Z}_+$ such that $S \le n$ and $S = n^{\Omega(1)}$ (each machine uses $O(S)$ space)

Output: matching in G

1 $\Delta \leftarrow n$, $V' \leftarrow V$, $M \leftarrow \emptyset$
2 **while** $\Delta \ge \frac{n}{S}(200\ln n)^{32}$ **do**
  `/* High-probability invariant: maximum degree in` $G[V']$ `bounded by` $\frac{3}{2}\Delta$ `*/`
3 $m \leftarrow \left\lfloor \sqrt{\frac{n\Delta}{S}} \right\rfloor$ `/* number of machines used */`
4 $\tau \leftarrow \left\lceil \frac{1}{16}\log_{120\alpha}(\Delta/m) \right\rceil$ `/* number of phases to emulate */`
5 Partition V' into m sets $V_1, \ldots, V_m$ by assigning each vertex independently uniformly at random.
6 **foreach** $i \in [m]$ **do in parallel**
7 If the number of edges in $G[V_i]$ is greater than $8S$, $V_i \leftarrow \emptyset$.
8 **for** $j \in [\tau]$ **do**
  $(V_i, M_{i,j}) \leftarrow$ `LocalPhase` $(i, G[V_i], \Delta/(2^{j-1}m))$
9 $V' \leftarrow \bigcup_{i=1}^{m} V_i$
10 $M \leftarrow M \cup \bigcup_{i=1}^{m}\bigcup_{j=1}^{\tau} M_{i,j}$
11 $\Delta \leftarrow \Delta/2^\tau$
12 Compute degrees of vertices V' in $G[V']$ and remove from V' vertices of degree at least 2Δ.
13 Directly simulate $M' \leftarrow$ `GlobalAlg`$(G[V'], 2\Delta)$, using $O(1)$ rounds per phase.
14 **return** $M \cup M'$

We present `ParallelAlg`, our parallel algorithm, as Algorithm 5. We write S to denote a parameter specifying the amount of space per machine. After the initialization of variables, the algorithm enters the main loop in Lines 2–11. The loop is executed as long as Δ, an approximate upper bound on the maximum degree in the remaining graph, is large enough. The loop implements the idea of running multiple iterations of `LocalPhase` on each induced subgraph in a random partition. At the beginning of the loop, the algorithm decides on m, the number of machines, and τ, the number of phases to be emulated. Then it creates a random partition of the current set of vertices that results in m induced subgraphs. Next for each subgraph, the algorithm first runs a security check that the set of edges fits onto a single machine (see Line 7). If it does not, which is highly unlikely, the entire subgraph is removed from the graph. Otherwise, the entire subgraph is sent to a single machine that runs τ consecutive iterations of `LocalPhase`. Then the vertices not removed in the executions of `LocalPhase` are collected for further computation and new matching edges are added to the matching being constructed. During the execution of the loop, the maximum degree in the graph induced by V', the set of vertices to be considered is bounded by $\frac{3}{2}\Delta$ with high probability. Once the loop finishes, we remove from the graph vertices of degree higher than 2Δ—there should be none—and we directly simulate `GlobalAlg` on the remaining graph, using $O(1)$ rounds per phase.

Concentration Inequality. We use the following version of the Chernoff bound that depends on an upper bound on the expectation of the underlying independent random variables. It can be shown by combining two applications of the more standard version.

Lemma 5.1 (Chernoff bound). *Let $X_1, \ldots, X_k$ be independently distributed random variables taking values in $[0, 1]$. Let $X \stackrel{\text{def}}{=} X_1 + \cdots + X_k$ and let $U \ge 0$ be an upper bound on the expectation of X, i.e., $\mathbb{E}[X] \le U$. For any $\delta \in [0, 1]$, $\Pr(|X - \mathbb{E}[X]| > \delta U) \le 2\exp(-\delta^2 U/3)$.*

5.1 Properties of Thresholds

Before we analyze the behavior of the algorithm, we observe that the value of $\frac{\Delta}{m}$ inside the main loop is at least polylogarithmic and that the same property holds for the rescaled threshold that is passed to `LocalPhase`.

Lemma 5.2. *Consider a single iteration of the main **while** loop of `ParallelAlg` (Lines 2–11). Let Δ and m be set as in this iteration. The following two properties hold:*

- $\Delta/m \ge (200\log n)^{16}$.
- *The threshold $\Delta/(2^{j-1}m)$ passed to `LocalPhase` in Line 8 is always at least $(\Delta/m)^{15/16} \ge 4000\mu_R^{-2}\ln^2 n$.*

Proof. Let τ be also as in this iteration of the loop. The smallest threshold passed to `LocalPhase` is $\Delta/(2^{\tau-1}m)$. Let $\lambda \stackrel{\text{def}}{=} S\Delta/n$, where S is the parameter to `ParallelAlg`. Due to the condition in Line 2, $\lambda \ge (200\ln n)^{32}$. Note that $\Delta = \lambda n/S$. Hence $m \le \sqrt{n\Delta/S} = \frac{n}{S}\sqrt{\lambda}$. This implies that $\Delta/m \ge \sqrt{\lambda} \ge (200\ln n)^{16}$, which proves the first claim. Due to the definition of τ,

$$2^{\tau-1} \le (120\alpha)^{\tau-1} \le (\Delta/m)^{1/16}.$$

This implies that

$$\begin{aligned}\Delta/(2^{\tau-1}m) &\ge (\Delta/m)^{15/16} \ge (200\ln n)^{15} \\ &> 4\cdot 10^{15}\cdot\ln^4 n = 4000\mu_R^{-2}\ln^2 n.\end{aligned}$$

□

We also observe that the probability of any set of vertices deleted by the security check in Line 7 of ParallelAlg is low as long as the maximum degree in the graph induced by the remaining vertices is bounded.

Lemma 5.3. *Consider a single iteration of the main* ***while*** *loop of* ParallelAlg *and let* Δ *and* V' *be as in that iteration. If the maximum degree in* $G[V']$ *is bounded by* $\frac{3}{2}\Delta$*, then the probability of any subset of vertices deleted in Line 7 is* n^{-8}*.*

Proof. Let m be as in the same iteration of the main loop of ParallelAlg. Consider a single vertex $v \in V'$. The expected number of v's neighbors assigned to the same subgraph is at most $\frac{3}{2}\Delta/m$. Recall that due to Lemma 5.2, $\frac{\Delta}{m} \geq 200 \ln n$. Since the assignment of vertices to machines is fully independent, by Lemma 5.1 (i.e., the Chernoff bound), the probability that v has more than $2\Delta/m$ neighbors is bounded by

$$2\exp\left(-\frac{1}{3}\cdot\left(\frac{1}{3}\right)^2\cdot\frac{3}{2}\cdot\frac{\Delta}{m}\right) \leq 2\exp\left(-\frac{1}{18}\cdot 200\ln n\right) \leq n^{-10}.$$

Therefore, by the union bound, with probability $1-n^{-9}$, no vertex has more than 2Δ neighbors in the same induced subgraph. As $|V'| \leq n$, the expected number of vertices in each set V_i constructed in the iteration of the main loop is at most $n/m \geq \Delta/m \geq 200\ln n$. What is the probability that $|V_i| > 2n/m$? By the independence of vertex assignments and Lemma 5.1, the probability of such event is at most

$$2\exp\left(-\frac{1}{3}\cdot\frac{n}{m}\right) \leq 2\exp\left(-\frac{1}{3}\cdot 200\ln n\right) \leq n^{-10}.$$

Again by the union bound, the event $|V_i| \leq 2n/m$, for all i simultaneously, occurs with probability at least $1-n^{-9}$. Combining both bounds, with probability at least $1-2n^{-9} \geq 1-n^{-8}$, all induced subgraphs have at most $2n/m$ vertices and the degree of every vertex is bounded by $2\Delta/m$. Hence the number of edges in one induced subgraph is at most $\frac{1}{2}\cdot\frac{2n}{m}\cdot\frac{2\Delta}{m} = \frac{2n\Delta}{m^2}$. By the definition of m and the setting of parameters in the algorithm, $m \geq \frac{1}{2}\sqrt{\frac{n\Delta}{S}}$, where S is the parameter to ParallelAlg. This implies that the number of edges is at most $2n\Delta/\left(\frac{1}{2}\sqrt{\frac{n\Delta}{S}}\right)^2 = 8S$ in every induced subgraph with probability $1-n^{-8}$, in which case no set V_i is deleted in Line 7 of ParallelAlg. □

5.2 Matching Size Analysis

The parallel algorithm runs multiple iterations of LocalPhase on each induced subgraph, without repartitioning. A single iteration on all subgraphs corresponds to running EmulatePhase once. We now show that in most cases, the global algorithm simulates EmulatePhase on a well behaved distribution with independently assigned vertices and all vertices distributed nearly uniformly conditioned on the configurations of the previously removed sets R_i, H_i, and F_i. We also show that the maximum degree in the remaining graph is likely to decrease gracefully during the process.

Lemma 5.4. *With probability at least* $1-n^{-3}$*:*

- *all parallel iterations of* LocalPhase *in* ParallelAlg *on each induced subgraph correspond to running* EmulatePhase *on independent and* $(200\ln n)^{-1}$*-near uniform distributions of assignments,*
- *the maximum degree of the graph induced by the remaining vertices after the* k*-th simulation of* EmulatePhase *is* $\frac{3}{2}\Delta/2^k$*.*

Proof. We first consider a single iteration of the main loop in ParallelAlg. Let Δ, τ, and m be set as in this iteration of the loop. For $j \in [\tau]$, let $\Delta_j \stackrel{\text{def}}{=} \Delta/(2^{j-1}m)$ be the threshold passed to LocalPhase for the j-th iteration of LocalPhase on each of the induced subgraphs. The parallel algorithm assigns vertices to subgraphs and then iteratively runs LocalPhase on each of them. In this analysis we ignore the exact assignment of vertices to subgraphs until they get removed as a member of one of sets R_i, H_i, or F_i. Instead we look at the conditional distribution on assignments given the configurations of sets R_i, H_i, and F_i removed in the previous iterations corresponding to EmulatePhase. We write $\mathcal{D}_j$, $1 \leq j \leq \tau$, to denote this distribution of assignments before the execution of j-th iteration of LocalPhase on the induced subgraphs, which corresponds to the j-th iteration of EmulatePhase for this iteration of the main loop of ParallelAlg. Additionally, we write $\mathcal{D}_{\tau+1}$ to denote the same distribution after the τ-th iteration, i.e., at the end of the execution of the parallel block in Lines 6–8 of ParallelAlg. Due to Lemma 4.2, the distributions of assignments are all independent. We define ϵ_j, $j \in [\tau+1]$, to be the minimum positive value such that $\mathcal{D}_j$ is ϵ_j-near uniform. Obviously, $\epsilon_1 = 0$, since the first distribution corresponds to a perfectly uniform assignment. We want to apply Lemma 4.3 inductively to bound the value of ϵ_{j+1} as a function of ϵ_j with high probability. The lemma lists two conditions: ϵ_j must be at most $(200\ln n)^{-1}$ and the threshold passed to EmulatePhase has to be at least $4000\mu_H^{-2}\ln^2 n$. The latter condition holds due to Lemma 5.2. Hence as long as ϵ_j is sufficiently small, Lemma 4.3 implies that with probability at least $1-n^{-4}$,

$$\epsilon_{j+1} \leq 60\alpha\left(\left(\frac{\Delta}{2^{\tau-1}m}\right)^{-1/4}+\epsilon_j\right) \leq 60\alpha\left(\left(\frac{\Delta}{m}\right)^{-15/64}+\epsilon_j\right),$$

and no high degree vertex survives in the remaining graph. One can easily show by induction that if this recursion is satisfied for all $1 \leq j \leq \tau$, then $\epsilon_j \leq (120\alpha)^{j-1}\cdot\left(\frac{\Delta}{m}\right)^{-15/64}$ for all $j \in [\tau+1]$. In particular, by the definition of τ and Lemma 5.2, for any $j \in [\tau]$,

$$\begin{aligned}\epsilon_j &\leq (120\alpha)^{\tau-1}\cdot\left(\frac{\Delta}{m}\right)^{-15/64}\\ &\leq \left(\frac{\Delta}{m}\right)^{1/16}\cdot\left(\frac{\Delta}{m}\right)^{-15/64} \leq \left(\frac{\Delta}{m}\right)^{-11/64} \leq (200\ln n)^{-1}.\end{aligned}$$

This implies that as long the unlikely events specified in Lemma 4.3 do not occur for any phase in any iteration of the main loop of ParallelAlg, we obtain the desired properties: all intermediate distributions of possible assignments are $(200\ln n)^{-1}$-near uniform and the maximum degree in the graph decreases at the expected rate. It remains to bound the probability of those unlikely events occurring for any phase. By the union bound, their total probability is at most $\log n \cdot n^{-4} \leq n^{-3}$. □

We now prove that the algorithm finds a large matching with constant probability.

Theorem 5.5. *Let M_{OPT} be an arbitrary maximum matching in a graph G. With $\Omega(1)$ probability,* `ParallelAlg` *constructs a matching of size $\Omega(|M_{\text{OPT}}|)$.*

Proof. By combining Lemma 5.3 and Lemma 5.4, we learn that with probability at least $1 - n \cdot n^{-8} - n^{-3} \geq 1 - 2n^{-3}$, we obtain a few useful properties. First, all relevant distributions corresponding to iterations of `EmulatePhase` are independent and $(200 \ln n)^{-1}$-near uniform. Second, the maximum degree in the graph induced by vertices still under consideration is bounded by $\frac{3}{2}\Delta$ before and after every simulated execution of `EmulatePhase`, where Δ is the corresponding. As a result, no vertex is deleted in Lines 7 or 12 due to the security checks.

We now use Lemma 4.1 to lower bound the expected size of the matching created in every `EmulatePhase` simulation. Let $\tau_\star$ be the number of phases we simulate this way. We have $\tau_\star \leq \log n$. Let $\mathbf{H}_j$, $\mathbf{F}_j$, and $\mathbf{M}_j$ be random variables equal to the total size of sets H_i, F_i, and M_i created in the j-th phase. If the corresponding distribution in the j-th phase is near uniform and the maximum is bounded, Lemma 4.1 yields

$$\mathbb{E}\left[\mathbf{H}_j + \mathbf{F}_j\right] \leq n^{-9} + 1200 \cdot \mathbb{E}\left[\mathbf{M}_j\right],$$

i.e.,

$$\mathbb{E}\left[\mathbf{M}_j\right] \geq \frac{1}{1200}\left(\mathbb{E}\left[\mathbf{H}_j + \mathbf{F}_j\right] - n^{-9}\right).$$

Overall, without the assumption that the conditions of Lemma 4.1 are always met, we obtain a lower bound

$$\sum_{j \in [\tau_\star]} \mathbb{E}\left[\mathbf{M}_j\right] \geq \sum_{j \in [\tau_\star]} \frac{1}{1200}\left(\mathbb{E}\left[\mathbf{H}_j + \mathbf{F}_j\right] - n^{-9}\right) - 2n^{-3} \cdot \frac{n}{2},$$

in which we consider the worst case scenario that we lose as much as $n/2$ edges in the size of the constructed matching when the unlikely negative events happen. `ParallelAlg` continues the construction of a matching by directly simulating the global algorithm. Let $\tau'_\star$ be the number of phases in that part of the algorithm. We define $\mathbf{H}'_j$, $\mathbf{F}'_j$, and $\mathbf{M}'_j$, for $j \in [\tau'_\star]$, to be random variables equal to the size of sets H, F, and $\widetilde{M}$ in `GlobalAlg` in the j-th phase of the simulation. By Lemma 3.3, we have

$$\sum_{j \in [\tau'_\star]} \mathbb{E}\left[\mathbf{M}'_j\right] \geq \sum_{j \in [\tau'_\star]} \frac{1}{50}\left(\mathbb{E}\left[\mathbf{H}'_j + \mathbf{F}'_j\right]\right).$$

By combining both bounds we obtain a lower bound on the size of the constructed matching. Let

$$\mathbf{M}_\star \stackrel{\text{def}}{=} \sum_{j \in [\tau_\star]} \mathbb{E}\left[\mathbf{M}_j\right] + \sum_{j \in [\tau'_\star]} \mathbb{E}\left[\mathbf{M}'_j\right]$$

be the expected matching size, and let

$$\mathbf{V}_\star \stackrel{\text{def}}{=} \sum_{j \in [\tau_\star]} \mathbb{E}\left[\mathbf{H}_j + \mathbf{F}_j\right] + \sum_{j \in [\tau'_\star]} \mathbb{E}\left[\mathbf{H}'_j + \mathbf{F}'_j\right].$$

We have

$$\mathbf{M}_\star \geq \frac{1}{1200}\mathbf{V}_\star - \frac{1}{n^2}.$$

Consider a maximum matching M_{OPT}. At the end of the algorithm, the graph is empty. The expected number of edges in M_{OPT} incident to a vertex in one of the reference sets is bounded by $|M_{\text{OPT}}| \cdot 2\mu_R \cdot \log n \leq 10^{-5}|M_{\text{OPT}}|$. The expected number of edges removed by the security checks is bounded by $\frac{n}{2} \cdot n^{-3}$. Hence the expected number of edges in M_{OPT} deleted as incident to vertices that are heavy or are friends is at least $(1 - 10^{-5})|M_{\text{OPT}}| - 1/(2n^2)$. Since we can assume without the loss of generality that the graph is non-empty, it is at least $\frac{1}{2}|M_{\text{OPT}}|$. Hence $\mathbf{V}_\star \geq \frac{1}{2}|M_{\text{OPT}}|$, and $\mathbf{M}_\star \geq \frac{1}{2400}|M_{\text{OPT}}| - \frac{1}{n^2}$. For sufficiently large n (say, $n \geq 50$), $\mathbf{M}_\star \geq \Omega(|M_{\text{OPT}}|)$ and by an averaging argument, `ParallelAlg` has to output an $O(1)$-multiplicative approximation to the maximum matching with $\Omega(1)$ probability. For smaller n, it is not difficult to show that at least one edge is output by the algorithm with constant probability as long as it is not empty. □

Finally, we want to argue that the above procedure can be used to compute $2 + \epsilon$ approximation to maximum matching at the cost of increasing the running time by a factor of $\log(1/\epsilon)$. The idea is to; execute algorithm `ParallelAlg` to compute constant approximate matching; remove this matching from the graph; and repeat.

Corollary 5.6. *Let M_{OPT} be an arbitrary maximum matching in a graph G. For any $\epsilon > 0$, executing* `ParallelAlg` *on G and removing a constructed matching repetitively, $O(\log(1/\epsilon))$ times, finds a multiplicative $(2+\epsilon)$-approximation to maximum matching, with $\Omega(1)$ probability.*

Proof. Assume that the `ParallelAlg` succeeds with probability p and computes c-approximate matching. Observe that each successful execution of `ParallelAlg` finds a matching M_c of size at least $\frac{1}{c}|M_{\text{OPT}}|$. Removal of M_c from the graph decreases the size of optimal matching by at least $\frac{1}{c}|M_{\text{OPT}}|$ and at most by $\frac{2}{c}|M_{\text{OPT}}|$, because each edge of M_c can be incident to at most two edges of M_{OPT}. Hence, when the size of the remaining matching drops to at most $\epsilon|M_{\text{OPT}}|$, we have an $2 + \epsilon$-multiplicative approximation to maximum matching constructed. The number t of successful applications of `ParallelAlg` need to satisfy.

$$\left(1 - \frac{1}{c}\right)^t \leq \epsilon.$$

This gives $t = O(\log(1/\epsilon))$. In $\lceil t/p \rceil = O(\log(1/\epsilon))$ executions, we have t successes with probability at least $1/2$ by the properties of the median of the binomial distribution. □

ACKNOWLEDGMENTS

We thank Sepehr Assadi, MohammadHossein Bateni, Aaron Bernstein, Mohsen Ghaffari, Vahab Mirrokni, Cameron Musco, Christopher Musco, Seth Pettie, Govind Ramnarayan, and Cliff Stein for helpful discussions, feedback and valuable comments. We also thank anonymous reviewers for their valuable feedback.

REFERENCES

[1] Kook Jin Ahn and Sudipto Guha. 2015. Access to Data and Number of Iterations: Dual Primal Algorithms for Maximum Matching under Resource Constraints. In *Proceedings of the 27th ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2015, Portland, OR, USA, June 13–15, 2015*. 202–211. https://doi.org/10.1145/2755573.2755586

[2] Noga Alon, László Babai, and Alon Itai. 1986. A Fast and Simple Randomized Parallel Algorithm for the Maximal Independent Set Problem. *Journal of Algorithms* 7, 4 (1986), 567–583. https://doi.org/10.1016/0196-6774(86)90019-2

[3] Alexandr Andoni, Aleksandar Nikolov, Krzysztof Onak, and Grigory Yaroslavtsev. 2014. Parallel algorithms for geometric graph problems. In *Proceedings of the 46th ACM Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31–June 3, 2014*. 574–583. https://doi.org/10.1145/2591796.2591805

[4] Sepehr Assadi. 2017. Simple Round Compression for Parallel Vertex Cover. *CoRR* abs/1709.04599 (September 2017). https://arxiv.org/abs/1709.04599

[5] Sepehr Assadi, MohammadHossein Bateni, Aaron Bernstein, Vahab S. Mirrokni, and Cliff Stein. 2017. Coresets Meet EDCS: Algorithms for Matching and Vertex Cover on Massive Graphs. *CoRR* abs/1711.03076 (2017). arXiv:1711.03076 http://arxiv.org/abs/1711.03076

[6] Sepehr Assadi and Sanjeev Khanna. 2017. Randomized Composable Coresets for Matching and Vertex Cover. In *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2017, Washington DC, USA, July 24–26, 2017*. 3–12. https://doi.org/10.1145/3087556.3087581

[7] Paul Beame and Johan Hastad. 1987. Optimal Bounds for Decision Problems on the CRCW PRAM. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC 1987)*. New York, NY, USA, 83–93. https://doi.org/10.1145/28395.28405

[8] Paul Beame, Paraschos Koutris, and Dan Suciu. 2013. Communication steps for parallel query processing. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA, June 22–27, 2013*. 273–284. https://doi.org/10.1145/2463664.2465224

[9] Paul Beame, Paraschos Koutris, and Dan Suciu. 2014. Skew in parallel query processing. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS'14, Snowbird, UT, USA, June 22–27, 2014*. 212–223. https://doi.org/10.1145/2594538.2594558

[10] Aaron Bernstein and Cliff Stein. 2015. Fully Dynamic Matching in Bipartite Graphs. In *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming, ICALP 2015, Kyoto, Japan, July 6–10, 2015, Proceedings, Part I*. 167–179. https://doi.org/10.1007/978-3-662-47672-7_14

[11] Aaron Bernstein and Cliff Stein. 2016. Faster Fully Dynamic Matchings with Small Approximation Ratios. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10–12, 2016*. 692–711. https://doi.org/10.1137/1.9781611974331.ch50

[12] Sayan Bhattacharya, Deeparnab Chakrabarty, and Monika Henzinger. 2017. Deterministic Fully Dynamic Approximate Vertex Cover and Fractional Matching in O(1) Amortized Update Time. In *Proceedings of the 19th International Conference on Integer Programming and Combinatorial Optimization, IPCO 2017, Waterloo, ON, Canada, June 26–28, 2017*. 86–98. https://doi.org/10.1007/978-3-319-59250-3_8

[13] Sayan Bhattacharya, Monika Henzinger, and Giuseppe F. Italiano. 2015. Deterministic Fully Dynamic Data Structures for Vertex Cover and Matching. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4–6, 2015*. 785–804. https://doi.org/10.1137/1.9781611973730.54

[14] Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. 2016. New deterministic approximation algorithms for fully dynamic matching. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18–21, 2016*. 398–411. https://doi.org/10.1145/2897518.2897568

[15] Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. 2017. Fully Dynamic Approximate Maximum Matching and Minimum Vertex Cover in $O(\log^3 n)$ Worst Case Update Time. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16–19*. 470–489. https://doi.org/10.1137/1.9781611974782.30

[16] Jeffrey Dean and Sanjay Ghemawat. 2004. MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation, Volume 6 (OSDI'04)*. USENIX Association, Berkeley, CA, USA, 10–10. http://dl.acm.org/citation.cfm?id=1251254.1251264

[17] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: Simplified Data Processing on Large Clusters. *Commununication of the ACM* 51, 1 (Jan. 2008), 107–113. https://doi.org/10.1145/1327452.1327492

[18] Jack Edmonds. 1965. Paths, Trees and Flowers. *Canadian Journal of Mathematics* (1965), 449–467.

[19] Jon Feldman, S. Muthukrishnan, Anastasios Sidiropoulos, Clifford Stein, and Zoya Svitkina. 2010. On distributing symmetric streaming computations. *ACM Transactions on Algorithms* 6, 4 (2010), 66:1–66:19. https://doi.org/10.1145/1824777.1824786

[20] Manuela Fischer and Mohsen Ghaffari. 2017. Deterministic Distributed Matching: Simpler, Faster, Better. *CoRR* abs/1703.00900 (2017). http://arxiv.org/abs/1703.00900

[21] Michael T. Goodrich, Nodari Sitchinava, and Qin Zhang. 2011. Sorting, searching, and simulation in the MapReduce framework. In *International Symposium on Algorithms and Computation*. Springer, 374–383.

[22] Michał Hańćkowiak, Michał Karoński, and Alessandro Panconesi. 2001. On the Distributed Complexity of Computing Maximal Matchings. *SIAM Journal on Discrete Mathematics* 15, 1 (2001), 41–57. https://doi.org/10.1137/S0895480100373121

[23] Michał Hańćkowiak, Michał Karoński, and Alessandro Panconesi. 1999. A Faster Distributed Algorithm for Computing Maximal Matchings Deterministically. In *Proceedings of the Eighteenth Annual ACM Symposium on Principles of Distributed Computing (PODC '99)*. ACM, New York, NY, USA, 219–228. https://doi.org/10.1145/301308.301360

[24] Zengfeng Huang, Božidar Radunović, Milan Vojnović, and Qin Zhang. 2015. Communication Complexity of Approximate Matching in Distributed Graphs. In *Proceedings of the 32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4–7, 2015, Garching, Germany*. 460–473. https://doi.org/10.4230/LIPIcs.STACS.2015.460

[25] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. 2007. Dryad: Distributed Data-parallel Programs from Sequential Building Blocks. *SIGOPS Operating Systems Review* 41, 3 (March 2007), 59–72. https://doi.org/10.1145/1272998.1273005

[26] Amos Israeli and Alon Itai. 1986. A Fast and Simple Randomized Parallel Algorithm for Maximal Matching. *Inform. Process. Lett.* 22, 2 (1986), 77–80. https://doi.org/10.1016/0020-0190(86)90144-4

[27] Amos Israeli and Yossi Shiloach. 1986. An Improved Parallel Algorithm for Maximal Matching. *Inform. Process. Lett.* 22, 2 (1986), 57–60. https://doi.org/10.1016/0020-0190(86)90141-9

[28] Michael Kapralov, Sanjeev Khanna, and Madhu Sudan. 2014. Approximating matching size from random streams. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5–7, 2014*. 734–751. https://doi.org/10.1137/1.9781611973402.55

[29] Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. 2010. A Model of Computation for MapReduce. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17–19, 2010*. 938–948. https://doi.org/10.1137/1.9781611973075.76

[30] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. 2006. The Price of Being Near-sighted. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithm (SODA 2006)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 980–989. http://dl.acm.org/citation.cfm?id=1109557.1109666

[31] Silvio Lattanzi, Benjamin Moseley, Siddharth Suri, and Sergei Vassilvitskii. 2011. Filtering: a method for solving graph problems in MapReduce. In *Proceedings of the 23rd Annual ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2011, San Jose, CA, USA, June 4–6, 2011*. 85–94. https://doi.org/10.1145/1989493.1989505

[32] Zvi Lotker, Boaz Patt-Shamir, and Seth Pettie. 2015. Improved Distributed Approximate Matching. *J. ACM* 62, 5, Article 38 (Nov. 2015), 17 pages. https://doi.org/10.1145/2786753

[33] Michael Luby. 1986. A Simple Parallel Algorithm for the Maximal Independent Set Problem. *SIAM J. Comput.* 15, 4 (1986), 1036–1053. https://doi.org/10.1137/0215074

[34] Andrew McGregor. 2005. Finding Graph Matchings in Data Streams. In *Approximation, Randomization and Combinatorial Optimization, Algorithms and Techniques, 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2005 and 9th InternationalWorkshop on Randomization and Computation, RANDOM 2005, Berkeley, CA, USA, August 22-24, 2005, Proceedings*. 170–181. https://doi.org/10.1007/11538462_15

[35] Krzysztof Onak and Ronitt Rubinfeld. 2010. Maintaining a large matching and a small vertex cover. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5–8 June 2010*. 457–464. https://doi.org/10.1145/1806689.1806753

[36] Michal Parnas and Dana Ron. 2007. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theoretical Computer Science* 381, 1-3 (2007), 183–196. https://doi.org/10.1016/j.tcs.2007.04.040

[37] Tim Roughgarden, Sergei Vassilvitskii, and Joshua R. Wang. 2016. Shuffles and Circuits: On Lower Bounds for Modern Parallel Computation. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2016, Asilomar State Beach/Pacific Grove, CA, USA, July 11–13, 2016*. 1–12. https://doi.org/10.1145/2935764.2935799

[38] Tom White. 2012. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc.

[39] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster Computing with Working Sets. In *2nd USENIX Workshop on Hot Topics in Cloud Computing, HotCloud'10, Boston, MA, USA, June 22*. https://www.usenix.org/conference/hotcloud-10/spark-cluster-computing-working-sets