Optimal Round and Sample-Size Complexity for Partitioning in Parallel Sorting

Wentao Yang*
Department of Computer Science
University of Illinois
Urbana-Champaign
Urbana, IL, USA
wentaoy2@illinois.edu

Vipul Harsh*
Department of Computer Science
University of Illinois
Urbana-Champaign
Urbana, IL, USA
vharsh2@illinois.edu

Edgar Solomonik
Department of Computer Science
University of Illinois
Urbana-Champaign
Urbana, IL, USA
solomon2@illinois.edu

ABSTRACT

State-of-the-art parallel sorting algorithms for distributed-memory architectures are based on computing a balanced partitioning via sampling and histogramming. By finding samples that partition the sorted keys into evenly-sized chunks, these algorithms minimize the number of communication rounds required. Histogramming (computing positions of samples) guides sampling, enabling a decrease in the overall number of samples collected. We derive lower and upper bounds on the number of sampling/histogramming rounds required to compute a balanced partitioning. We improve on prior results to demonstrate that when using p processors, $O(\log^* p)$ rounds with $O(p/\log^* p)$ samples per round suffice. We match that with a lower bound that shows that any algorithm with $O(p)$ samples per round requires at least $\Omega(\log^* p)$ rounds. Additionally, we prove the $\Omega(p \log p)$ samples lower bound for one round, thus proving that existing one round algorithms: sample sort, AMS sort [2] and HSS [16] have optimal sample size complexity. To derive the lower bound, we propose a hard randomized input distribution and apply classical results from the distribution theory of runs.

CCS CONCEPTS

• **Theory of computation → Massively parallel algorithms**.

KEYWORDS

parallel algorithms, parallel sorting, communication cost, communication lower bounds

ACM Reference Format:
Wentao Yang, Vipul Harsh, and Edgar Solomonik. 2023. Optimal Round and Sample-Size Complexity for Partitioning in Parallel Sorting. In *Proceedings of the 35th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '23), June 17–19, 2023, Orlando, FL, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3558481.3591076

*Both authors contributed equally

SPAA '23, June 17–19, 2023, Orlando, FL, USA

ACM ISBN 978-1-4503-9545-8/23/06…$15.00
https://doi.org/10.1145/3558481.3591076

1 INTRODUCTION

1.1 Background

Sorting a distributed array (a common problem and widely used primitive) is challenging due to the communication overhead needed to infer a load-balanced partition of the global order of the data items, especially for sorting large scale data. Modern parallel sorting algorithms minimize data movement by first computing an approximately balanced partitioning of the global order, then sending data directly to the destination processor. Among the simplest such algorithms is *sample sort* [13], which infers the partitioning by collecting a sample of data items of each processor and by picking evenly spaced samples as splitters that are close to quantiles in the global order. Sample sort achieves a balanced partitioning with p processors given a sample size of $\Theta(p \log p)$, but such a sample can become expensive to store/process for large p, such as in a work by Cheng et al. [11], where p is often in the order of hundred thousands. Multi-round sampling can be used to lower the total sample size, by *histogramming* [18, 23] samples and drawing subsequent samples from a subset of the global input range. Specifically, a *histogram* of a sample gives the rank in the global order of each item in the sample. For example, the *Histogram sort with sampling* algorithm has been shown to require $O(\log \log p)$ rounds of $O(p)$ samples per round to achieve a balanced partitioning.

There is a fundamental tradeoff between the total sample size and the total of number of sample-histogram rounds (which can be performed with 2 BSP supersteps). Less rounds are desired, but additional rounds permit more selective sampling and consequently less communication. The exact relationship of the tradeoff, as well as the optimal round complexity and sample complexity remain open problems.

1.2 Our contribution

In this paper, we aim to address these problems. Our contribution is twofold. First, we provide a tighter analysis of Histogram sort with sampling [16]. We demonstrate that with a total sample size complexity of $O(p)$ using $O(\log^* p)$ rounds[1] of sampling followed by histogramming (Section 2) suffices to achieve a balanced partition with high probability. We refer to the adjusted sampling/histogramming stage of the HSS algorithm as *Histogram Partitioning*. Second, we show that the sample size and round complexity of the approach are optimal by proving (Section 3) that with $O(p)$ samples per round, $\Omega(\log^* p)$ rounds are necessary for any

[1] We use the following definition of $\log^* x$ for any positive real number x: $\log^* x = 0$ if $x \leq 1$, $\log^* x = 1 + \log^*(\log x)$ if $x > 1$.

randomized algorithm to compute a balanced partitioning. To prove this lower bound, we design a difficult input distribution and apply Yao's principle, making use of results from distribution theory of success runs to derive the result. A corollary of our lower bounds is that, since any one round algorithm requires $\Omega(p \log p)$ samples - Sample sort, HSS with one round, AMS-sort [2] are essentially optimal in terms of sample size complexity. Figure 1 illustrates the various tradeoff points between the number of sample-histogram rounds vs the total sample size and our lower bounds.

1.3 Related work

In data-heavy large scale settings, often it's infeasible to move data multiple times. Hence, large scale parallel sorting algorithms [2, 16, 23, 24] first determine a set of $p-1$ splitters to split the data into p buckets (p is the total number of processors). The set of splitters are broadcast to all processors and each processor sends keys falling between the $(i-1)^{th}$ and the i^{th} splitter to i^{th} processor. The choice of the splitters directly determines how load balanced the final data is across processors. Often, algorithms [2, 16] provide an extra guarantee that each processor would end up with no more than $\frac{N(1+\epsilon)}{p}$ keys where N is the total number of keys across all processors. Past works on parallel sorting have extensively studied algorithms that efficiently compute the set of splitters (also referred to as a partition) that guarantee $(1+\epsilon)$ load imbalance. The data-partitioning algorithm is also the focus of this paper. The final part of parallel sorting is the data exchange phase where each processor sends data to each other based on the splitters. Efficient data-exchange algorithms are outside the scope of this paper. We refer the reader to [2] for a study.

Sample sort [13], an early parallel sorting algorithm achieves $(1+\epsilon)$-balanced partitioning by sampling keys either randomly [8] or evenly across all processors [21], requiring a total sample size of $O(\frac{p \log p}{\epsilon^2})$. Histogramming is the process of carrying out a reduction across all processors to obtain the ranks of all sampled keys. Histogramming allows the partitioning algorithm to make a more informed choice and reduces the sample size requirements. AMS-sort with 1 sample-histogram round requires $O(p(\log p + \frac{1}{\epsilon}))$ samples. We note that the sample size is directly proportional to the communication cost of the data partitioning step [16], hence algorithms aim to minimize the sample size.

One could achieve a significantly lower sample size complexity by repeated rounds of sampling followed by histogramming [16, 24]. Histogram sort with Sampling (HSS) [16] achieves a sample size complexity of $O(p \log\log p)$ with $O(\log\log p)$ rounds of sampling/histogramming, while Hyksort with a slightly different sampling algorithm achieves a sample size complexity of $O(p \log p)$ [16]. Note that in this work, we focus on sample sort and histogram sort-like algorithms, which require a reduction of samples to one processor and a broadcast of data from that processor to others. We refer the reader to [16] for a detailed review of these algorithms.

The optimal communication cost for parallel sorting algorithms in a general BSP model has been previously studied by Goodrich [14]. Goodrich [14] gives an optimal algorithm that requires $O(\frac{\log N}{\log h+1})$ rounds with each processor sending and receiving at most $h = \Theta(\frac{N}{p})$ items per round. Goodrich also proves a tight round lower bound $\Omega(\frac{\log N}{\log h+1})$. Our lower bound analysis applies in a more restricted setting, which we discuss in Section 1.5. This theoretical setting characterizes partition-based parallel sorting algorithms that perform well on current architectures (HSS and HykSort). Partition-based algorithms have the advantage of moving data (aside from samples) only once or twice, unlike merge-based algorithms such as the one employed by Goodrich.

To the best of our knowledge, our analysis is the first to apply the distribution theory of runs [22], which quantify the frequency of sequences of successes in a series of random trials, to analysis of parallel sorting/partitioning. This connection may be employed for new or simplified analysis of other sampling-based partitioning algorithms for parallel sorting or related problems. Further information on the topic may be found in prior surveys of topics related to the distribution theory of runs [3].

1.4 Problem Statement

Let $A(1), \ldots, A(N)$ be an input sequence of keys (without duplicates) distributed across p processing elements, such that each processor has $\frac{N}{p}$ keys. We assume that the keys could be of any data type, so that the algorithm is purely comparison-based. Let $R(k)$ denote the rank of key k in the array A and let $I(r)$ denote the key of rank r. The goal of parallel sorting is to sort the input keys, so that the i^{th} processor owns the i^{th} subsequence of the sorted keys $I(1), \ldots, I(N)$ and that each subsequence consists of at most $(1+\epsilon)\frac{N}{p}$ keys. The goal of the partitioning algorithm is to determine $p-1$ splitter keys $s_1, s_2, \ldots, s_{p-1}$ which divide the input range into p buckets. We refer to the set of splitter keys as a partition. We wish to achieve a well-balanced partition so that the size of each bucket is small. We choose s_0 and s_p such that $R(s_0) = 0$ and $R(s_p) = N$ for notational convenience.

Though we focus on the parallel partitioning problem in this paper, we make references to the parallel sorting problem since parallel partitioning is a crucial part of parallel sorting- if the input data has been partitioned, each processor can simply sort locally which solves the parallel sorting problem.

Definition 1. *A partition $(s_0 \le s_1 \le s_2 \ldots \le s_p)$ is $(1+\epsilon)$-balanced if $R(s_{i+1}) - R(s_i) \le (1+\epsilon)\frac{N}{p} \quad \forall i \in \{0, \ldots, p-1\}$.*

Table 1 defines further notation used in the algorithm and the lower bound proofs.

algorithm	p	number of processors sorting keys
	N	number of keys to sort
	$A(j)$	the j^{th} input key, A is the set of all keys
	$R(i)$	initial index of the key ranked i^{th} globally
	$I(r)$	key with rank r in the overall global order
	W_i	the set of undetermined splitters are the i^{th} round
lower bound	C	the number of sub-intervals in each interval (same as the number of contiguous sequences at the start of the first round)
	T	the number of rounds in the algorithm
	D_i	bound on the number of contiguous sequences at round i
	Q_i	the number of intervals in each contiguous sequence at round i
	K_i	the number of samples available for each interval on average at round i

Table 1: Notation used in this paper.

1.5 Cost and Execution Model

In this work, we restrict our attention to sample-sort and histogram-sort like algorithms in the BSP model. This allows us to prove a different lower bound on round complexity than the previously mentioned result by Goodrich in the general setting of parallel sorting in the BSP model [14]. Our lower bound also applies to the Map-Reduce model, in which only one-to-all and all-to-one communications are allowed.

We analyze partitioning algorithms based on the total number of samples collected and the number of sampling/histogramming rounds. These bounds naturally correspond to round/synchronization complexity and communication cost in multiple models. One such model is the bulk synchronous parallel (BSP) [25]. In the BSP model, algorithms proceed in *supersteps*[2]. In each superstep,

- each processor spends a fixed amount of time performing local computation,
- each processor sends/receives message up to a given total size (h) to/from other processors.

Our algorithm and lower bound for the cost of partitioning apply in a BSP setting with $h = p$, but also require that only $O(p)$ distinct keys are collected/communicated at each step. To align our setting with that of Goodrich [14], we would need $h = \Theta(p) = \Theta(n/p)$, in which case Goodrich's algorithm requires a constant number of rounds, as it leverages communication/exchange $O(hp)$ total keys per round instead of $O(h)$. We further note that our lower bound applies to the weaker problem of finding a $(1+\epsilon)$-balanced partitioning as opposed to complete parallel sorting.

Our results also lead to complexity bounds for algorithms in the massively parallel computing (MPC) model [19]. The MPC model is very similar to the BSP model, but it focuses on bounding the number of rounds in a distributed algorithm, given a limit on the amount of memory. In partitioning algorithms $\Theta(\frac{N}{p} + p)$ memory is typically used to store local data and the partition. Our algorithm requires $O(\frac{N}{p} + p)$ memory and $O(\log^* p)$ rounds, and our tight lower bound still holds in the setting given similar restrictions. This constitutes a reduction in memory footprint by $\Theta(\log p)$ with respect to sample sort. Figure 1 provides a summary of the lower and upper bounds we introduce, as well as past work (Sample sort, HSS [16] and AMS-sort [2]).

Collective operations: In our algorithm, we make use of collective operations that are extensively used in parallel algorithms. We refer to figure 1 of [9] for a quick overview of some common collective operations. Specifically, in the BSP model, broadcasts of $O(p)$ elements and reductions of $O(p)$ elements from each process, are efficiently done via scatter or reduce-scatter, respectively, followed by all-gather, both of which have communication cost $O(p)$ and require 1 BSP superstep.

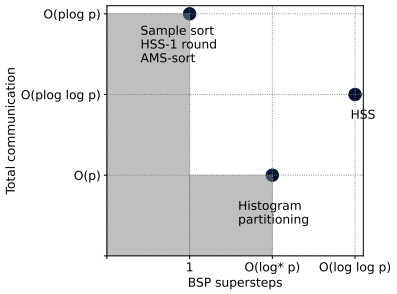

Figure 1: Complexity of histogram/sample-sort approaches in BSP supersteps and total communication. Histogram partitioning is the sampling/histogramming approach introduced and analyzed in this paper. The shaded region represents our lower bounds, no algorithm based on global sampling can be strictly inside the region.

2 SAMPLE ROUND COMPLEXITY UPPER BOUND AND FAST ALGORITHM

In this section, we review the HSS algorithm [16], expressed for a more general sample size. We then analyze the histogramming/sampling stage with a different choice of sampling ratios, which yields our Histogram Partitioning approach. For any constant $\epsilon > 0$, when seeking a $(1+\epsilon)$-balanced partitioning, the new analysis shows that $O(\log^* p)$ rounds and $O(p)$ communication volume suffice (opposed to $O(\log \log p)$ rounds and $O(p \log \log p)$ communication volume, as analyzed in [16]).

2.1 Histogram Sort with Sampling

We first describe the Histogram Sort with Sampling (HSS) algorithm from [16] for a general sampling ratio. In each round, the HSS algorithm computes a histogram of a sample of keys. A histogram of keys $a_1, \ldots, a_q$ is given by their global ranks $R(a_1), \ldots, R(a_q)$. The global ranks are computed by summing (reduction) of the local ranks of the keys $a_1, \ldots, a_q$ within the set of keys assigned to each processor. Each histogram serves to narrow down the range of keys out of which the subsequent samples are collected, with the aim of finally finding keys $a_1, \ldots, a_{p-1}$, whose global ranks give an accurate (load-balanced) splitting, namely $R(a_i) \in [i\frac{N}{p}, i(1+\epsilon)\frac{N}{p}]$. We refer to such an a_i as a 'determined' splitter.

In general, given two keys a, b with $a < b$, we define an interval $[a, b]$ as the set of keys in the input sequence that are greater than a and smaller than b. HSS maintains bounds for each of the $p-1$ splitters, which are successively improved. In the i^{th} round the bounds for splitter j are $[L_i(j), U_i(j)]$, with $L_1(j) = I(1)$, $U_1(j) = I(N)$ for all $j \in \{1, \ldots, p-1\}$. The set of undetermined splitters is at round i is W_i. The set of keys from which HSS collects a sample is given by the union of the bounds for all splitters which the algorithm has not yet determined. For a choice of sampling ratio ψ_i, the HSS algorithm computes the following steps at the i^{th} round (the pseudocode is also given in the Appendix as Algorithm 1):

[2]To clarify the notation, a round of sampling and broadcasting consists of two BSP supersteps.

- if all $p-1$ splitters have been determined, broadcast them, redistribute keys so that processor i receives keys between the i^{th} and $(i+1)^{th}$ splitters, sort locally, and terminate, otherwise,
- each processor collects a local sample by sampling each local key in $\gamma_i = \bigcup_{j \in W_i}[L_j, U_j]$ with probability ψ_i,
- the combined sample $a_1, \ldots, a_\eta$ is gathered globally, (e.g., by gathering all sample keys to one processor then broadcasting to all processors),
- each processor computes a local histogram of the sample,
- a global histogram $H = \{h_1, \ldots, h_\eta\}$, $h_j = R(a_j)$ is computed by reduction of local histograms,
- if any $H \ni h_j \in [\eta\frac{N}{p}, \eta(1+\epsilon)\frac{N}{p}]$, the corresponding sample a_j is a determined η^{th} splitter,
- for all undetermined splitters, $\eta \in W_{i+1}$, update the splitter bounds using the new histogram,

$$L_{i+1}(\eta) = \max\Big(L_i(\eta), \max_{h_j \in H, h_j < \eta\frac{N}{p}}(h_j)\Big) \quad \text{and}$$

$$U_{i+1}(\eta) = \min\Big(U_i(\eta), \min_{h_j \in H, h_j > \eta\frac{N}{p}(1+\epsilon)}(h_j)\Big),$$

- broadcast updates to the splitter bounds, namely the change to W_{i+1}, $W_i \setminus W_{i+1}$ and $[L_j, U_j]$ for all $j \in W_{i+1}$.

In [16], the sampling ratio at each round is determined by seeking a total sample of a fixed size ($O(p)$) at each step. In this work, we consider a constant $\epsilon \leq 1$ and set $\psi_i = \frac{p^2}{|W_i|N\log^* p}$. Since γ_i is composed of $|W_i|$ undetermined intervals as well as (due to $\epsilon \leq 1$) at most 1 determined interval before and after each undetermined interval, $|\gamma_i \cap A| \leq 3|W_i|N/p$. Hence, the expected number of samples in the i^{th} round is $\psi_i|\gamma_i \cap A| \leq 3\psi_i N|W_i|/p = \frac{3p}{\log^* p}$. This choice allows us to obtain a total communication volume of $O(p)$, since $O(p)$ communication suffices for both sampling and collecting updated splitter information on all processors. In the next subsection, we show that w.h.p. $O(\log^* p)$ rounds of HSS suffice with this choice of sample ratio. If aiming to achieve a small (non-constant) ϵ, another $O(\log(\frac{1}{\epsilon}))$ histogramming rounds with $O(p)$ communication volume per round are required, following the approach/analysis in [16].

2.2 New Running Time Upper Bound

We show that the modified HSS algorithm (which we refer to as Histogram Partitioning), finishes in $O(\log^* p)$ rounds and requires $O(p)$ samples in total (stated in Theorem 6). Our analysis is based on the intuition that the algorithm can be interpreted as a randomized string cutting process. We can think of the global input of keys (in sorted order) as a string of length N. Sampling a random key of rank r is equivalent to cutting the string at the r^{th} of N points. Each substring that is long enough yields an independent partitioning subproblem. Every Histogram Partitioning round cuts each substring in as many random points as the number of samples selected from that subrange.

Our analysis is based on 2 phases. In the first phase, there are still few substrings, so the sampling density is low. In the second phase, many splitters have already been determined, so the total length of all substrings is relatively small. We now define these phases formally and show that both complete in $O(\log^* p)$ rounds.

We split the iterations of Algorithm 2.1 into two phases. In the first phase, we show the partitioning algorithm makes progress so that $O\Big(\frac{p}{\log^* p}\Big)$ new splitters are determined in each iteration (Lemma 2). In the second phase, in each iteration the number of undetermined splitters goes down quickly according to the $\log^*$ function (Lemma 4).

- Phase 1: Iterations where the number of undetermined splitters exceeds $\frac{p}{\log^* p}$ before the start of the iteration.
- Phase 2: Iterations where the number of undetermined splitters is at most $\frac{p}{\log^* p}$ before the start of the iteration.

Since the total number of samples in each round is fixed ($\frac{3p}{\log^* p}$ in expectation), we only need to bound the number of iterations for a complete analysis. We show that both phases of the algorithm finish in $O(\log^* p)$ iterations with high probability (w.h.p.).

Our analysis keeps track of the of the size of the domain, γ_j, of input keys that are sampled from at the j^{th} step. The size of γ_j decreases whenever adjacent splitters are determined. Hence, if most splitters have been determined, the length of the input cannot be much larger than the set of intervals corresponding to undetermined splitters.

We first derive a lower bound for the probability of a previously undetermined splitter being determined in round j. Note that the sampling method ensures that sampling in disjoint intervals is independent.

Lemma 2. *If at the start of the j^{th} step of the Histogram Partitioning algorithm, $k = |W_j| \geq \frac{p}{\log^* p}$ splitters are undetermined (Phase 1), then the probability that a given undetermined splitter is determined during the j^{th} step is at least $\frac{p}{2k\log^* p}$.*

Proof. The j^{th} round of Histogram Partitioning samples elements in γ_j with probability $\psi_j = p^2/(|W_j|N\log^* p)$. The probability that any previously undetermined splitter is determined is then given by the probability that at least one of the $\frac{N}{p}$ keys in the corresponding interval is sampled,

$$\begin{aligned}1 - \Big(1-\psi_j\Big)^{\frac{N}{p}} &= 1 - \Big(1 - \frac{p^2}{|W_j|N\log^* p}\Big)^{\frac{N}{p}} = 1 - \Big(1 - \frac{p^2}{kN\log^* p}\Big)^{\frac{N}{p}} \\ &\geq 1 - e^{\frac{-p}{k\log^* p}} \geq \frac{p}{k\log^* p} - \frac{1}{2}\Big(\frac{p}{k\log^* p}\Big)^2 \\ &\geq \frac{p}{k\log^* p} - \frac{1}{2}\Big(\frac{p}{k\log^* p}\Big) = \frac{p}{2k\log^* p}.\end{aligned}$$

The last inequality uses that $\frac{p}{k\log^* p} \leq 1$, which follows from $k \geq \frac{p}{\log^* p}$, since the algorithm is in Phase 1. □

Lemma 2 allows us to bound the number of iterations in phase 1.

Lemma 3. *Phase 1 of the algorithm finishes (fewer than $\frac{p}{\log^* p}$ samples remain undetermined) after $O(\log^* p)$ iterations w.h.p..*

Proof. We use a multiplicative Chernoff's bound to show that in each iteration, at least $\frac{p}{3\log^* p}$ new splitters are determined w.h.p.. Specifically, let X be the number of determined splitters in the j^{th}

round of phase 1. We have $X = X_1 + X_2 \ldots + X_k$ where X_i is an indicator random variable with $X_i = 1$ if the i^{th} undetermined splitter is determined in the j^{th} round. Note that the sampling method ensures that sampling in disjoint intervals are independent. Hence, all X_i's are independent of each other. From Lemma 2, we have $\mathbb{E}[X] \geq k\frac{p}{2k\log^* p} = \frac{p}{2\log^* p}$, where k is the number of undetermined splitters before the j^{th} round. Using multiplicative Chernoff bounds, we get

$$\Pr[X < \frac{p}{3\log^* p}] = \Pr\left[X < \frac{p}{2\log^* p}\left(1 - \frac{1}{3}\right)\right] \leq e^{-\frac{p}{2\log^* p}\frac{1}{3^2}},$$

which gives the high probability bound.

□

We now bound the number of iterations needed for phase 2 to find all remaining undetermined splitters. In phase 2, the number of undetermined splitters are less than or equal to $\frac{p}{\log^* p}$. We show that the number of undetermined splitters remaining after a round decreases exponentially with the gap from $\frac{p}{\log^* p}$.

Lemma 4. *If the number of undetermined splitters is $k \leq \frac{p}{t\log^* p}$ with $1 \leq t \leq \log\log p$ at the start of the j^{th} iteration of the Histogram Partitioning algorithm, then the number of undetermined splitters after the j^{th} iteration is at most $k/2^t$, w.h.p..*

Proof. Let X_i be an indicator random variable, with $X_i = 1$ denoting that the ith splitter remains undetermined after the iteration. All X_i's are independent. To show the result in the theorem, we seek to bound $X = X_1 + \cdots + X_k$. The probability that the i^{th} previously undetermined splitter remains undetermined is

$$\Pr[X_i = 1] = \left(1 - \psi_j\right)^{\frac{N}{p}} = \left(1 - \frac{p^2}{kN\log^* p}\right)^{\frac{N}{p}} \leq \left(1 - \frac{pt}{N}\right)^{\frac{N}{p}} \leq e^{-t}.$$

Hence, $\mathbb{E}[X] \leq ke^{-t}$. Since our sampling strategy is such that sampling within splitter intervals is independent, we can apply a multiplicative Chernoff bound to bound X with high probability.

Using Chernoff's bound[3], we get

$$\Pr[X \geq k2^{-t}] = \Pr\left[X \geq \mathbb{E}[X]\left(1 + \frac{k2^{-t} - \mathbb{E}[X]}{\mathbb{E}[X]}\right)\right] \leq e^{-\frac{B^2\mathbb{E}[X]}{2}},$$

where $B = \frac{k2^{-t} - \mathbb{E}[X]}{\mathbb{E}[X]}$. Since, $ke^{-t}/\mathbb{E}[X] \geq 1$,

$$B^2 \geq ((e/2)^t - 1)^2 \geq 1/9.$$

Thus, the probability that X is less than the bound is small, since $e^{-\frac{B^2\mathbb{E}[X]}{2}} = O(e^{-\mathbb{E}[X]}) = O(e^{-p/(\log p\log^* p)})$. □

Lemma 4 shows that the number of undetermined splitters goes down rapidly from $\frac{p}{t\log^* p}$ to $\frac{p}{t2^t\log^* p}$, which immediately leads us to an upper bound for the number of iterations in phase 2 to achieve all splitters.

Lemma 5. *Phase 2 of the Histogram Partitioning algorithm (iterations after fewer than $\frac{p}{\log^* p}$ splitters remain undetermined) finishes (all splitters are determined) after $O(\log^* p)$ iterations w.h.p..*

[3]We use the multiplicative Chernoff bound, $P[X > (1+\epsilon)\mathbb{E}[X]] \leq e^{\frac{-\epsilon^2\mathbb{E}[X]}{2}}$.

Proof. The number of undetermined splitters at the beginning of phase 2 is at most $\frac{p}{\log^* p}$. By Lemma 4, after k iterations, at most[4] $\frac{p}{(2\uparrow\uparrow k)\log^* p}$ undetermined splitters remain w.h.p.. Therefore, after $O(\log^* p)$ iterations in phase 2, all splitters are determined w.h.p.. If the number of undetermined splitters becomes smaller than $p/(\log p\log^* p)$ at any point, then all splitters are determined after one more round using the analysis of HSS for one round [16].

□

We can now combine the iteration bounds for the two phases to conclude with our main theorem.

Theorem 6. *The Histogram Partitioning algorithm with $\epsilon = 1$ finds a 2-balanced partition in $O(\log^* p)$ BSP steps (and an additional $O(\log(\frac{1}{\epsilon}))$ steps for any $\epsilon > 0$) and $O(p)$ total communication w.h.p..*

Proof. Combining Lemma 3 and Lemma 5, we can assert that Histogram Partitioning determines all splitters after $\log^* p$ iterations, w.h.p.. The communication in each round of the algorithm consists of gathering a sample of size $O(\frac{p}{\log^* p})$ and computing a histogram via a reduction of size $O(\frac{p}{\log^* p})$. The bounds defining γ_j may be updated locally (redundantly on all processors) or with $O(\frac{p}{\log^* p})$ communication of distinct updated bounds. Consequently, the communication cost of the algorithm in the BSP model is $O(p)$. For any $\epsilon > 0$, refer to the analysis in [16].

□

3 PARTITIONING ROUND COMPLEXITY LOWER BOUND

In this section, we prove that any randomized algorithm for finding a 2-balanced partition, that communicates $O(p)$ total keys per round, requires $\Omega(\log^* p)$ rounds to achieve a constant probability of success. Our result implies that any algorithm needs at least $\Omega(\log^* p)$ rounds to find a Δ-balanced partition with $O(p)$ samples per round, thus giving us the $\Omega(\log^* p)$ sample round complexity lower bound for partitioning with $O(p)$ samples per round.

The proof is structured as follows (see Figure 3 in the Appendix for an overview),

- we use Yao's principle to argue that the probability that for the worst-case input, any randomized algorithm cannot find a 2-balanced partition is greater than the probability that the best deterministic algorithm cannot find a 2-balanced partition for any randomized input distribution;
- we propose a randomized input distribution that we show is hard for any deterministic partitioning algorithm;
- to prove the hardness of our proposed input distribution, we invoke a theorem from the distribution theory of runs to upper bound the amount of progress that can be made in each round by any deterministic algorithm;
- we use an inductive argument to give a concrete lower bound for the probability that a certain number of splitters remain undetermined in each round given $T = \frac{\log^* p}{6}$ total rounds;

[4]$\uparrow\uparrow$ denotes tetration, the inverse operation of $\log^*$.

- finally, we use an union bound on the probabilities for each round to argue that with high probability, 2-balanced partition cannot be determined for large enough p.

Our lower bound proof can also characterize all deterministic algorithms that use the information of $O(p)$ samples (for any way of choosing the samples) at each round. We restrict our attention to asymptotic analysis and assume the number of keys (N) is large.

3.1 Complexity Bounds from Sampling Bounds

We now show that it suffices to consider deterministic histogram-based algorithms on a particular distribution of inputs.

3.1.1 Reduction to Histogram-based Algorithms. We start by showing that, having restricted to communication of $O(r)$ keys per round, comparison-based algorithms cannot perform significantly better than histogram-based algorithms (for some choice of keys to histogram at each round). First, we show that obtaining a 2-balanced partition is essentially as hard as the global-partitioning obtained by Histogram sort defined below.

Definition 7. *A partition $(s_0 \leq s_1 \leq s_2 ... \leq s_p)$ is $(1+\epsilon)$-globally-balanced if $R(s_i) \in [i\frac{N}{p}, (i+\epsilon)\frac{N}{p}]$ $\forall i \in \{1, \ldots, p-1\}$.*

Note the difference from a $(1+\epsilon)$-balanced partition defined in Definition 1. A $(1+\epsilon)$-globally-balanced partition is also a $(1+\epsilon)$-balanced partition.

Lemma 8. *Given a distribution of inputs, let $T(p)$ be the expected round complexity of the best deterministic comparison-based 2-balanced partitioning BSP algorithm that communicates at most $r \geq p$ keys in each round. Then, there exists a 2-globally-balanced partitioning algorithm with expected round complexity $T(2p)$ that communicates $2r$ keys per round.*

Proof. We may obtain a 2-globally-balanced partitioning by performing 2-balanced partitioning with $2p$ virtual processors, with each of p processors simulating 2 virtual processors. In a 2-balanced partitioning with $2p$ processors, no processor may be assigned more than $\frac{N}{p}$ elements, hence the $2p-1$ splitters defining the partitioning must include a splitter from each of the p intervals, which together yield a 2-globally-balanced partitioning. Execution with $2p$ processors can be done in $T(2p)$ rounds and at most $2r$ communication per round (ensuring the theorem assumption, $2r \geq 2p$), yielding a round complexity of $T(2p)$. □

Next, we show that, in our setting, any algorithm that communicates a given set of keys might as well compute their histogram (global positions), and hence it suffices to consider histogram-based algorithms to derive a round complexity lower bound. We note that a sample-histogram round requires 2 BSP supersteps (one to collect the sample and one to reduce the histogram).

Lemma 9. *Given a distribution of inputs, the round complexity of any deterministic comparison-based 2-globally-balanced partitioning BSP algorithm that communicates $O(p)$ keys in each step is no better than the number of sample-histogram rounds needed by a Histogram sort that computes the position of the same $O(p)$ keys in each round.*

Proof. Consider the set of comparisons between keys computed at each round by the given algorithm. Aside from comparisons of pairs of keys assigned to the same processor initially, these comparisons must involve only previously communicated keys. Histogram sort computes the position in the global order of all communicated keys, which amounts to performing comparisons between all keys and the communicated keys. Consequently, the given algorithm can infer that a 2-globally-balanced partitioning has been found only if at a given round, the previously communicated keys include a key from each interval, in which case Histogram sort also completes. □

3.1.2 Reduction to Sampling Algorithms. Yao's principle allows us to obtain a lower bound on the number of rounds needed by any randomized partitioning algorithm for the worst-case input, by reduction to the expected round complexity of the best deterministic algorithm for a randomized distribution of inputs. We first state Yao's principle in general form then describe our case as a corollary.

Lemma 10 (Yao's principle). *Let F_T denote the space of possible inputs for a problem, $\mathcal{R}_T$ denote the distribution of the outcomes of any randomized algorithm, A_T denote the space of possible deterministic algorithms for the problem, and $\mathcal{D}_T$ denote the input distribution. By Yao's principle, for any cost function Θ that is a real-valued measure of a deterministic algorithm on an input, we have*[5]

$$\max_{f \in F_T} \mathbb{E}_{R \leftarrow \mathcal{R}_T}[\Theta(R, f)] \geq \min_{A \in A_T} \mathbb{E}_{f \leftarrow \mathcal{D}_T}[\Theta(A, f)]. \quad (1)$$

Corollary 11. *Consider the problem of finding 2-partition via a comparison-based BSP algorithm that communicates $O(p)$ total keys per round. Consider a given input distribution $\tilde{\mathcal{D}}_T$. Also, let $X(R, f)$ define the event that R cannot find 2-partition for f in T rounds, and let $X(A, f)$ define the event that A cannot find 2-partition for f in T rounds. Then, we have*

$$\max_{f \in F_T} \Pr_{R \leftarrow \mathcal{R}_T}[X(R, f)] \geq \min_{A \in A_T} \Pr_{f \leftarrow \tilde{\mathcal{D}}_T}[X(A, f)]. \quad (2)$$

Proof. After setting cost of achieving $X(R, f)$ and $X(A, f)$ as 0, and 1 otherwise, the corollary trivially follows from Lemma 10. □

As a result, we only need to show that $\min_{A \in A_T} \Pr_{f \leftarrow \tilde{\mathcal{D}}_T}[X(A, f)]$, which is a probabilistic lower bound for any deterministic algorithm on our input distribution, is large enough. Such a bound gives us a probabilistic lower bound for the original problem.

We choose the input distribution $\tilde{\mathcal{D}}_T$ by partitioning the global order of keys into C parts, each containing $\frac{p}{C}$ consecutive intervals, and assigning each processor a single random subinterval from each part. The distribution (Figure 2) is thus a 3-level partition of input into C parts, containing a total of $C \cdot \frac{p}{C}$ intervals, and $C \cdot \frac{p}{C} \cdot C$ subintervals. We describe the process to generate this distribution in detail below (illustrated in Figure 2).

Construction 12 (input distribution). *Given a sorted input of N keys, we follow the steps below to assign the global input to each processor, which gives us the input distribution:*

(1) divide the global input evenly into p intervals;
(2) divide the p intervals into C parts with each part containing $\frac{p}{C}$ intervals (a part is just a sequence of intervals);

[5] Here $\mathbb{E}_{R \leftarrow \mathcal{R}_T}$ denotes the expectation with random variable R taken over distribution $\mathcal{R}_T$.

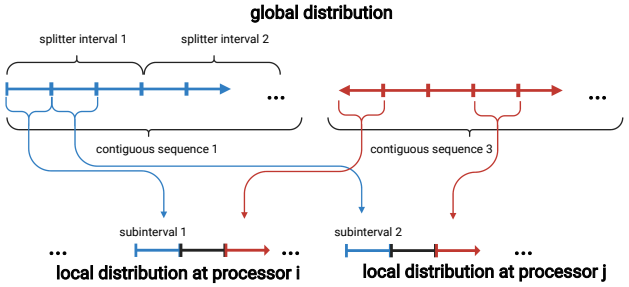

Figure 2: The input distribution we use to prove hardness. The distribution is obtained by assuming a global sorted order of input keys (at the top) and then constructing the local distribution at each process from the global order (below).

(3) divide each interval evenly into C subintervals so that each part has p subintervals;
(4) for every part, assign exactly one subinterval randomly to each processor.

The merit of the chosen distribution is that each processor owns keys from a single random interval in that part. Consequently, if the part is unsampled (the global positions of all keys within it are unknown), a given processor may be able to tell which of its keys belong to the part, but cannot tell which interval in the part they belong to. More generally, we refer to sequences of unsampled intervals within a single part as a **contiguous sequence**.

Definition 13 (contiguous sequence). *A contiguous sequence is defined to be a contiguous sequence of unsampled intervals of input in the global order such that any processor contains at most one subinterval from the continguous sequence in its local input.*

We note that in the first round, every part is a contiguous sequence of $\frac{p}{C}$ intervals (p subintervals). The next lemma shows that our distribution ensures that sampling in different contiguous intervals is independent.

Lemma 14. *At any round, for any deterministic histogram-based algorithm, subintervals in each contiguous sequence have the same probability to be sampled by the algorithm. Consequently, intervals in each contiguous sequence have the same probability to be sampled.*

Proof. Given the input distribution above, for any contiguous sequence, because each processor contains at most one subinterval and no matter how we interchange these subintervals among processors they would still appear in the same position of the their respective local orderings, for any deterministic algorithm, processors cannot distinguish one subinterval from other subintervals in any assigned contiguous sequence. Moreover, since subintervals in each contiguous sequence are distributed randomly, they are sampled with the same probability. Therefore, intervals in each contiguous sequence are sampled with the same probability. □

The lemma above essentially claims that throughout T rounds, for any contiguous sequence, processors cannot infer any information globally about that contiguous sequence from its local input. Thus, for any deterministic algorithm, at each round, processors can only make the decision of whether to sample a random subinterval in a given contiguous sequence. Therefore, any deterministic algorithm is equivalent to some sampling strategy of distributing samples for different contiguous sequences.

3.2 Bound for One Round/BSP Superstep

We now derive a sample-size lower bound for 2-balanced partitioning for single round partitioning algorithms, which yields a lower bound on the sample size needed by the sample sort algorithm. We first characterize the probability that a given interval in a contiguous sequence is unsampled, then use this to bound the sample size needed to ensure all intervals in one or more contiguous sequences are sampled. We restrict our analysis of algorithms to what contiguous sequences they choose to sample from, as we have established (1) in Lemma 9, that no comparison-based algorithm that communicates $O(p)$ total keys per round can complete in fewer steps than the minimum rounds needed by a histogram-based scheme, and (2) in Lemma 14 that with our input distribution, processors cannot distinguish intervals within the same contiguous sequence.

Lemma 15. *Given KQ samples (across all processors) from a contiguous sequence with Q intervals, the probability that any given interval in the contiguous sequence is unsampled is at least e^{-4K}, if $\frac{K}{C} \le 1/4$ and $Q \ge 2$.*

Proof. By Lemma 14, the sampling strategy of any deterministic algorithm for one contiguous sequence is equivalent to random sampling of subintervals in the sequence. Let X_i be the event that at least one subinterval in interval i is sampled. X_i does not occur if and only if each subinterval from interval i falls in the set of $CQ - KQ$ unsampled subintervals from the CQ total subintervals in the contiguous sequence (we assume, without loss of generality, that all samples are chosen from distinct subintervals). We then compute the probability $1-\Pr[X_i]$ that the C subintervals in interval i are unsampled, which can be formulated as

$$\begin{aligned} 1 - \Pr[X_i] &= \prod_{i=0}^{C-1} \frac{CQ - KQ - i}{CQ - i} \ge \left(\frac{CQ - KQ - C}{CQ - C}\right)^C \\ &= \left(1 - \frac{K}{C}\frac{Q}{Q-1}\right)^C \ge \left(1 - \frac{2K}{C}\right)^C \ge e^{\frac{-4KC}{C}} = e^{-4K}, \end{aligned} \tag{3}$$

where we replace each term in the product with the lower bound and use $e^{-x} \le 1 - x + \frac{x^2}{2} \le 1 - \frac{x}{2}$ if $0 \le x \le 1$. □

Having obtained a bound on the probability of an interval remaining unsampled, we now bound the sample size needed to sample all unsampled intervals with a given probability.

Lemma 16. *Given D contiguous sequences, each containing Q unsampled intervals, any deterministic algorithm that samples all unsampled intervals in one round, with probability at least ρ, needs to sample QKD keys in total, where $K \ge \frac{1}{4}\log\left(\frac{DQ}{\log(\frac{1}{\rho})}\right)$ assuming $\frac{K}{C} \le 1/4$ and $Q \ge 2$.*

Proof. We first characterize the case of one contiguous sequence, that is, $D = 1$. Let X_i be the event that interval i is sampled, and we want to compute an upper bound for $\Pr[X_1, X_2, ..., X_Q]$. We

first show that

$$\forall i, \Pr[X_i|X_1, ..., X_{i-1}] \leq \Pr[X_i], \tag{4}$$

which can be easily verified since the probability that for some interval i^{th}, j^{th} subinterval in the interval is in one of the $CQ-KQ-(j-1)$ unsampled positions out of $CQ-(j-1)$ possible positions is less than the probability that the j^{th} subinterval is in one of the $CQ-KQ-(j-1)$ unsampled positions out of $CQ-(j-1)-(i-1)$ possible positions since at least $i-1$ samples must be in $X_1...X_{i-1}$.

Then, from Lemma 15, we have

$$\begin{aligned}\Pr[X_1, X_2...X_Q] &= \Pr[X_1]P[X_2|X_1]...\Pr[X_Q|X_1, ...X_{Q-1}] \\ &\leq \Pr[X_1]\Pr[X_2]...\Pr[X_Q] \leq \left(1-e^{-4K}\right)^Q,\end{aligned} \tag{5}$$

for one contiguous sequence.

For D contiguous sequences, the distribution of samples among contiguous sequences must satisfy $Q\sum_{i=1}^{D}\kappa_i = QKD$, where we sample $\kappa_i Q$ samples from the i^{th} contiguous sequence (note that the total number of samples is QKD). The optimal sampling strategy is then given by the solution to the optimization problem given by maximizing the probability of all intervals being sampled,

$$\max_{K_1,...,K_D} \prod_{i=1}^{D} \Pr_i[X_1, ..., X_Q|\kappa_i] \quad \text{s.t.} \quad \sum_{i=1}^{D}\kappa_i = KD, \text{ and } \forall i, \kappa_i \geq 0. \tag{6}$$

We upper bound the maximum of the above problem, by using our analysis of a single contiguous sequence (Eq. 5), which gives an upper bound of the above objective function of $\prod_{i=1}^{D}\left(1-e^{-4\kappa_i}\right)^Q$. By quantifying when this upper bound can reach ρ, we obtain a lower bound on sample size. By standard application of the method of Lagrange multipliers, the above upper bound is maximized when all $\kappa_i = K$. So, if the algorithm samples all unsampled intervals with probability ρ, then it must be that

$$\rho \leq \left(1-e^{-4K}\right)^{DQ} \leq e^{-DQe^{-4K}}, \tag{7}$$

which implies

$$K \geq \frac{1}{4}\log\left(\frac{DQ}{\log(\frac{1}{\rho})}\right). \tag{8}$$

□

With the above lemma, we now bound the sample-size neeeded to achieve a good partitioning in a single round (sample complexity of sample sort).

Theorem 17. *Any randomized algorithm that achieves 2-balanced partition, with probability greater than ρ, in a single round, requires $\Omega(p\log p)$ samples for large enough p.*

Proof. By Lemma 8, if there exists a single round 2-balanced partitioning algorithm with $r = O(p\log p)$ sample complexity for all p, then a single round algorithm with the same complexity also exists for 2-globally-balanced-partitioning. Further, by Lemma 9, to obtain a round lower-bound, it suffices to consider algorithms that sample/histogram a set of keys in each round. Finally, by Yao's principle (Corollary 11), it suffices to consider the round complexity of the best deterministic algorithm on the input distribution defined in Construction 12. By Lemma 16, for any ρ, for large enough p, C, Q (it suffices to let $C = Q = \sqrt{p}$), the number of samples needed to find a suitable 2-globally-balanced partition is

$$K \geq \frac{1}{4}\log\left(\frac{p}{\log(\frac{1}{\rho})}\right). \tag{9}$$

So, the algorithm needs at least $K = \Omega(\log(p))$, or $r = \Omega(p\log(p))$ samples in total to achieve 2-globally-balanced partition in one round. By the above reductions, this result implies that any single round randomized algorithm for 2-balanced partitioning communicates $\Omega(p\log p)$ total keys. □

3.3 Bound for $\Omega(\log^*(p))$ Rounds/BSP Supersteps

We now consider algorithms with multiple sample-histogram rounds, demonstrating that any BSP algorithm that communicates $O(p)$ total keys per round, requires $\Omega(\log^* p)$ rounds. To derive our lower bound, we make use of a result from the distribution theory of runs (A.M. Mood, 1940 [22]). This result allows us to bound how many sequences of subintervals of a certain size remain unsampled in a contiguous sequence after a round of sampling.

Definition 18. *Let $n^{(k)} = n\cdot(n-1)\cdot(n-2)\ldots(n-k+1)$, $n^{(0)} = 1$.*

Lemma 19 (distribution theory of runs). *Consider random arrangements of m element of two kinds, for example m_1 a's and m_2 b's such that $m_1 + m_2 = m$. Let r_{1i} denote the total number of runs of a's of length i. Let $s_{1k} = \sum_{i=k}^{m_1} r_{1i}$ denote the total number of runs of a's of length at least k. We have $\mathbb{E}[s_{1k}] = \frac{(m_2+1)m_1^{(k)}}{m^{(k)}}$ and $\mathrm{Var}[s_{1k}] = \frac{(m_2+1)^{(2)}m_1^{(2k)}}{m^{(2k)}} + \frac{(m_2+1)m_1^{(k)}}{m^{(k)}}\left(1-\frac{(m_2+1)m_1^{(k)}}{m^{(k)}}\right)$.*

Proof. Refer to equation *(3.13), (3.15)* in *The Distribution Theory of Runs* [22] by *A. M. Mood.* □

Lemma 19 immediately allows us to bound the number of subsequences of unsampled intervals after a sample-histogram round on a given contiguous sequence.

Corollary 20. *After K_iQ_i random subintervals are sampled from a contiguous sequence containing CQ_i subintervals, the number of maximal subsequences of unsampled subintervals of size at least n, $\hat{X}_{i+1}$, satisfies $\mathbb{E}[\hat{X}_{i+1}] = \frac{(K_iQ_i+1)(CQ_i-K_iQ_i)^{(n)}}{(CQ_i)^{(n)}}$ and $\mathrm{Var}[\hat{X}_{i+1}] = \frac{(K_iQ_i+1)^{(2)}((C-K_i)Q_i)^{(2n)}}{(CQ_i)^{(2n)}} + \mathbb{E}[\hat{X}_{i+1}](1-\mathbb{E}[\hat{X}_{i+1}])$.*

Proof. This result follows directly from Lemma 19, since we can treat each subinterval as an element and whether a subinterval is sampled as whether a element b appears in the arrangement, and the problem of characterizing $\hat{X}_{i+1}$ can be thought of as the problem of characterizing the number of runs of a's of size at least n in the random arrangements of CQ_i elements with $(CQ_i - K_iQ_i)$ a's and K_iQ_i b's. □

We now apply this bound in a setting with many contiguous subsequences. We aim to bound the number of contiguous subsequences at the i^{th} step (random variable denoted by X_i).

Definition 21. *Let $\alpha_i(K_i) = \frac{K_iQ_i+1}{e^{4K_i(Q_{i+1}+1)}}$.*

Lemma 22. *Consider D_i contiguous sequences, such that each contiguous sequence contains Q_i intervals, and a total of at most $K_iD_iQ_i$ subintervals are sampled from these contiguous sequences (randomly within each contiguous sequence), if $\frac{Q_{i+1}}{Q_i} \le \frac{1}{2}$ and $\frac{K_i}{C} \le \frac{1}{4}$, the number of maximal subsequences of unsampled intervals of size at least Q_{i+1} (for any distribution of samples across the D_i contiguous sequences), X_{i+1}, satisfies*

$$\mathbb{E}[X_{i+1}] \ge D_i\alpha_i(K_i). \tag{10}$$

Further, the probability that X_{i+1} is smaller than half of the expected value is bounded as follows,

$$\Pr[X_{i+1} \le \frac{1}{2}D_i\alpha_i(K_i)] \le \frac{4}{D_i\alpha_i(K_i)}. \tag{11}$$

Proof. Let κ_jQ_i be the number of samples from contiguous sequence j, with $\sum_{j=1}^{D_i}\kappa_j = D_iK_i$. Then, by Lemma 20, the number of maximal subsequences in contiguous sequence j of unsampled subintervals of size at least n satisfies $\mathbb{E}[\hat{X}_{i+1}(\kappa_j)] = \frac{(\kappa_jQ_i+1)(CQ_i-\kappa_jQ_i)^{(n)}}{(CQ_i)^{(n)}}$ and $\mathrm{Var}[\hat{X}_{i+1}(\kappa_j)] = \frac{(\kappa_jQ_i+1)^{(2)}((C-\kappa_j)Q_i)^{(2n)}}{(CQ_i)^{(2n)}} + \mathbb{E}[\hat{X}_{i+1}](1-\mathbb{E}[\hat{X}_{i+1}])$. We pick $n = C(Q_{i+1}+1)$, which ensures that the sequence of subintervals contains the entirety of Q_{i+1} intervals, and hence no subintervals in these intervals are sampled. Further, $\mathbb{E}[\hat{X}_{i+1}] \ge \alpha_i(\kappa_j)$ since, by similar analysis as in the one-round case,

$$\begin{aligned}\frac{\mathbb{E}[\hat{X}_{i+1}(\kappa_j)]}{\kappa_jQ_i+1} &= \prod_{j=0}^{n-1}\frac{CQ_i-\kappa_jQ_i-j}{CQ_i-j}\\ \ge \left(1-\frac{\kappa_jQ_i}{CQ_i-n}\right)^n &\ge \left(1-\frac{\kappa_jQ_i}{CQ_i-CQ_{i+1}}\right)^n\\ \ge \left(1-\frac{\kappa_jQ_i}{CQ_i-CQ_i/2}\right)^n &= \left(1-\frac{2\kappa_j}{C}\right)^n \ge e^{\frac{-4\kappa_jn}{C}} = e^{-4\kappa_j(Q_{i+1}+1)}.\end{aligned}$$

Since sampling is independent across all D_i contiguous sequences (since they are maximal, they must be disjoint), the total number of maximal subsequences of unsampled subintervals of size at least n, X_{i+1}, satisfies

$$\mathbb{E}[X_{i+1}] = \sum_{j=1}^{D_i}\mathbb{E}[\hat{X}_{i+1}(\kappa_j)] \ge \sum_{j=1}^{D_i}\alpha_i(\kappa_j).$$

Since, $\sum_{j=1}^{D_i}\kappa_j = D_iK_i$ and $\alpha_i'(x) < 0$ for $x > 1$, as

$$\begin{aligned}\alpha_i'(x) &= e^{-4x(Q_{i+1}+1)}(-4Q_i(Q_{i+1}+1)x + Q_i - 4(Q_{i+1}-1)\\ &\le -e^{-4x(Q_{i+1}+1)}(4Q_i(Q_{i+1}+1)),\end{aligned}$$

$\mathbb{E}[X_{i+1}]$ is minimized when $\kappa_j = K_i$, so $\mathbb{E}[X_{i+1}] \ge D_i\alpha_i(K_i)$.

To quantify the variance, we observe that for all $m, k \le n$,

$$n^{(2k)} \le \left(n^{(k)}\right)^2 \text{ and } \frac{(n-m)^{(2k)}}{n^{(2k)}} \le \left(\frac{(n-m)^{(k)}}{n^{(k)}}\right)^2. \tag{12}$$

Using the above inequalities and Corollary 20, the variance for one contiguous sequence is

$$\begin{aligned}\mathrm{Var}[\hat{X}_{i+1}(\kappa_i)] &= \mathbb{E}[\hat{X}_{i+1}(\kappa_i)]\\ &\quad + \left(\frac{(K_iQ_i+1)^{(2)}((C-K_i)Q_i)^{(2n)}}{(CQ_i)^{(2n)}} - \mathbb{E}^2[\hat{X}_{i+1}(\kappa_i)]\right)\\ &\le \mathbb{E}[\hat{X}_{i+1}(\kappa_i)].\end{aligned} \tag{13}$$

Further, across D_i sequences, using independence, we have

$$\begin{aligned}\mathrm{Var}[X_{i+1}] = \mathrm{Var}\left[\sum_{j=1}^{D_i}\hat{X}_{(i+1)}(\kappa_j)\right] &= \sum_{j=1}^{D_i}\mathrm{Var}[\hat{X}_{(i+1)}(\kappa_j)]\\ \le \sum_{j=1}^{D_i}\mathbb{E}[\hat{X}_{(i+1)}(\kappa_j)]] &\le \mathbb{E}[X_{(i+1)}].\end{aligned} \tag{14}$$

We now obtain the desired bound on deviation from expectation by using Cantelli's inequality,

$$\Pr[X_{i+1} \le \mathbb{E}[X_{i+1}] - \lambda] \le \frac{\mathrm{Var}[X_{i+1}]}{\mathrm{Var}[X_{i+1}] + \lambda^2}. \tag{15}$$

Substituting $\lambda = \frac{\mathbb{E}[X_{i+1}]}{2}$, we get

$$\begin{aligned}\Pr[X_{i+1} \le \frac{1}{2}D_i\alpha_i(K_i)] &\le \Pr[X_{i+1} \le \frac{1}{2}\mathbb{E}[X_{i+1}]]\\ \le \frac{\mathrm{Var}[X_{i+1}]}{\mathrm{Var}[X_{i+1}] + \mathbb{E}^2[X_{i+1}]/4} &\le \frac{\mathbb{E}[X_{i+1}]}{\mathbb{E}[X_{i+1}] + \mathbb{E}^2[X_{i+1}]/4}\\ = \frac{4}{4+\mathbb{E}[X_{i+1}]} \le \frac{4}{\mathbb{E}[X_{i+1}]} &\le \frac{4}{\mathbb{E}[X_{i+1}|\forall j, \kappa_j = K_i]} = \frac{4}{D_i\alpha_i(K_i)}.\end{aligned}$$

□

Theorem 23. *Any randomized algorithm, given $O(p)$ samples for each round, requires $T = \Omega(\log^*(p))$ number of rounds to achieve 2-balanced partitioning with probability greater than $1/2$, for large enough p.*

Proof. We show that any histogram-based sorting algorithm requires more than $T = \frac{\log^*(p)}{6}$ rounds to achieve a constant probability of success with p samples per round[6]. By the same argument as in the proof of Theorem 17, this bound also extends to the round complexity of any randomized algorithm that communicates a total of $O(p)$ keys per round, completing the proof. Our argument proceeds by induction on the round number i. We show that after the i^{th} sample-histogram round that, with probability at least $1 - \sum_{j=1}^{i}\frac{4}{D_i\alpha_i(K_i)}$, there are at least D_{i+1} contiguous sequences of

$$Q_{i+1} = \lfloor e^{T-i+1}\rfloor \tag{16}$$

(unsampled) intervals left, where $D_1 = C = \frac{p}{4e^T}$ and $D_{i+1} = \frac{D_i\alpha_i(K_i)}{2}$. We choose K_i such that $K_iD_iQ_i = p$ for all i. We conservatively assume that exactly D_i contiguous sequences are left before the i^{th} round with the above probability as having more than D_i contiguous sequences left can only worsen the complexity of the algorithm.

For $i = 0$ (before the first round), the inductive assumption holds since there are C contiguous (unsampled) sequences each

[6]Note that increasing the sample per round by a constant factor to $(p\alpha)$ per round can lead to at most a α factor decrease in the number of rounds for the lower bound.

containing $\frac{p}{C} = 4e^T > Q_1$ intervals. For the inductive step, given that prior to the $(i+1)^{th}$ sample-histogram, (assuming $K_i \le C/4$, which we prove next), we apply Lemma 22 to lower bound the number of unsampled contiguous sequences for the next round with high probability, namely, we obtain

$$\Pr[X_{i+1} \ge \frac{1}{2} D_i \alpha_i(K_i)] \ge \left(1 - \frac{4}{D_i \alpha_i(K_i)}\right)\left(1 - \sum_{j=1}^{i-1} \frac{4}{D_j \alpha_j(K_j)}\right) \ge 1 - \sum_{j=1}^{i} \frac{4}{D_j \alpha_j(K_j)}.$$

To complete the inductive argument, it then suffices to prove $K_i \le \frac{C}{4}$, given our inductive assumption.

We unravel the inductive relationship to bound K_i and to obtain the final bound on the probability after T steps. Assume, without loss of generality, that there are exactly p samples per round, then the number of samples per unsampled interval at round 1 is $K_1 = 1$. Further, at round $i+1$,

$$\begin{aligned} K_{i+1} &= \frac{p}{D_{i+1} Q_{i+1}} = \frac{2pe^{4K_i(Q_{i+1}+1)}}{D_i(K_i Q_i + 1)} Q_{i+1} \\ &\le \frac{2pe^{4K_i(Q_{i+1}+1)}}{D_i K_i Q_i} Q_{i+1} = 2e^{4K_i(Q_{i+1}+1)} Q_{i+1} \\ &\le M(T) e^{K_i M(T)}, \quad \text{where} \quad M(T) = 4(e^T + 1). \end{aligned} \tag{17}$$

Let

$$R_i = M(T) e^{M(T) e^{M(T) e^{\cdots^{M(T)}}}} \Big\} i \text{ powers.}$$

For large enough T, we have

$$\begin{aligned} \log^*(R_i) &= 1 + \log^*(\log R_i) = 1 + \log^*(\log M(T) + X_{i-1}) \\ &\le 1 + \log^*(X_{i-1}^2) < 1 + \log^*(e^{X_{i-1}}) \\ &\le 2 + \log^*(X_{i-1}) \le 2i + \log^*(M(T)) \\ &\le 2T + 1 + \log^*(2T) \le 3T. \end{aligned} \tag{18}$$

Note that[7] $\log^*(x) = 1 + \log^*(\log x)$ by definition. Therefore, since $\log^*(z) = y$ if and only if $z = e \uparrow\uparrow y$,

$$K_i < R_i = e \uparrow\uparrow \log^*(R_i) < e \uparrow\uparrow 3T. \tag{19}$$

Since, $T = \frac{\log^*(p)}{6}$, we have that

$$K_i < e \uparrow\uparrow 3T < e \uparrow\uparrow (\frac{1}{2}\log^*(p)) \le e \uparrow\uparrow (\log^*(p) - 2) = e \uparrow\uparrow (\log^*(\log^2(p))) = \log\log(p).$$

Since, $C = \frac{p}{4e^T}$ and $\frac{p}{e^T} > \log\log p$ for sufficiently large p, we have shown that $K_i < \frac{C}{4}$. Further, since $K_i D_i Q_i = p$ for all i, we have that

$$D_T = \frac{K_1 D_1 Q_1}{K_T Q_T} > \frac{C}{e \uparrow\uparrow 3T} > \frac{C}{\log\log p}. \tag{20}$$

[7] $\uparrow\uparrow$ denotes tetration and $\log^*$ denotes super-logarithm which is the inverse operation of tetration.

Since $D_T > \frac{C}{\log\log p} \ge 1$ for large enough p, it suffices to apply our inductive bound on probability,

$$\Pr[X_T \ge 1] \ge \Pr[X_T \ge \frac{C}{2\log\log p}] \ge \Pr[X_T \ge \frac{1}{2} D_{T-1} \alpha_{T-1}(K_{T-1})] \ge 1 - \sum_{j=1}^{T-1} \frac{4}{D_j \alpha_j(K_j)} \ge 1 - T\frac{2}{D_T} \ge 1 - \frac{T\log\log p}{2C} \ge \frac{1}{2},$$

for sufficiently large $p, C \ge T\log\log p$, which completes the proof. □

Note that we have proved a slightly stronger theorem than what we need to show optimality of the new algorithm. We have shown a lower bound of $\Omega(\log^* p)$ rounds when the algorithm samples $O(p\log^*(p))$ keys in total, instead of $O(p)$.

4 DISCUSSION AND RELATED WORK

Parallel sorting is an extensively studied algorithm, both in the experimental and theoretical contexts.

Multiple levels of partitioning and data movement: To alleviate the bottleneck due to the data partitioning step or in the final data-exchange (where $O(p^2)$ messages have to be exchanged), some algorithms [2, 16, 24] resort to a 2-level sorting algorithm where the algorithm first groups together several processors into a "processor-group" and first executes parallel sorting at the level of processor-groups and then within each processor group. In this context, the Histogram partitioning algorithm may be applied to efficiently partition keys across and within processor-groups.

Merge-based sorting networks: Classical results, such as by Batcher [4], Cole [12] and AKS [1] use sorting networks and optimize the depth of the network. Our paper deals with large scale parallel sorting which are *partitioning-based*, and move data only once, while typical sorting-networks naively imply many rounds of interprocessor communication. Leighton [20] presented tight lower bounds for sorting networks, but no such bound is known for partitioning-based algorithms.

Experimental validation: The HSS algorithm [16] has been shown to outperform other state-of-the-art parallel sorting implementations, as well as to achieve good scalability and partitioning balance. We note that Histogram partitioning is faster than HSS [16] by a factor of $\Omega(\log^* p)$ and while the sampling ratio suggested by our work is more efficient, in a practical setting it is unlikely to result in a large improvement over HSS.

Optimal sample complexity: Note that in Theorem 23, we have also given a formula to compute the lower bound for K_i (the number of samples at each round) and the total number of samples to achieve 2-balanced partitioning for any randomized algorithm for every constant round $T = 2, 3, 4\ldots$ similar to Theorem 17. Thus these results may be derived easily in future works.

Related partitioning problems: The partitioning problem we consider also provides a parallel method for computing an approximate ranking of a sequence [17]. In contrast to [17], our lower bounds assume all-to-one comparisons with each sample, as opposed to minimizing pairwise comparisons.

Communication-efficient parallel sorting algorithms have been the subject of many prior studies. For example [5], study sorting algorithms with asymmetric read and write costs, while [6, 7] study

cache oblivious sorting algorithms. We believe our input distribution and the technique to applying distribution theory of runs can be a valuable tool for analyzing new parallel sorting algorithms, as well as techniques for other problems such as approximate ranking, *semisort* [15], and *quantile computation* [10].

5 CONCLUSION

Data partitioning is a crucial building block of parallel sorting with several past approaches. Via new theoretical analysis, we introduce a histogramming strategy that achieves a balanced partitioning with minimal sample volume and round complexity ($O(p/\log^* p)$ samples per round, $O(\log^* p)$ rounds and $O(p)$ total samples). To show optimality, we present a lower bound ($\Omega(\log^* p)$) on the number of rounds required for any randomized algorithm to achieve a 2-partition with $O(p)$ samples per round. This analysis provides an asymptotically tight analysis of randomized histogramming, a widely used and practical parallel sorting technique.

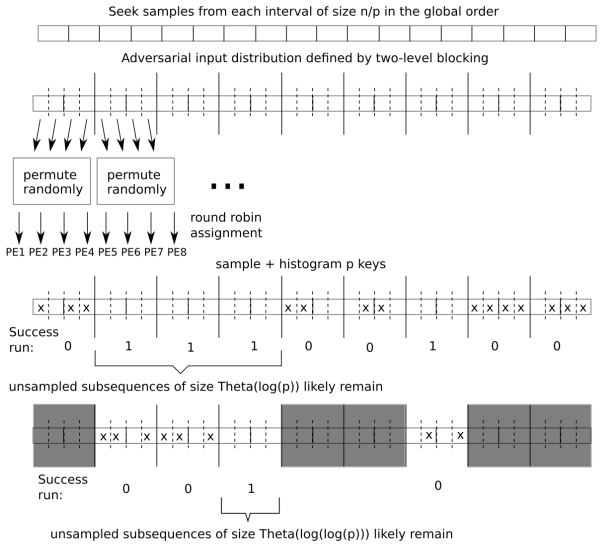

Figure 3: Overview of the construction of the input distribution used in our proof as well as the connection between rounds of sampling/histogramming and success runs.

REFERENCES

[1] Miklós Ajtai, János Komlós, and Endre Szemerédi. An O(n log n) sorting network. In *Proceedings of the 15th annual ACM Symposium on Theory of computing*, pages 1–9. ACM, 1983.

[2] Michael Axtmann, Timo Bingmann, Peter Sanders, and Christian Schulz. Practical massively parallel sorting. In *Proceedings of the 27th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 13–23. ACM, 2015.

[3] Narayanaswamy Balakrishnan and Markos V Koutras. *Runs and scans with applications*. John Wiley & Sons, 2011.

[4] Kenneth E Batcher. Sorting networks and their applications. In *Proceedings of the April 30–May 2, 1968, spring joint computer conference*, pages 307–314. ACM, 1968.

[5] Guy E. Blelloch, Jeremy T. Fineman, Phillip B. Gibbons, Yan Gu, and Julian Shun. Sorting with asymmetric read and write costs. In *Proceedings of the 27th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '15, pages 1–12, New York, NY, USA, 2015. ACM.

[6] Guy E. Blelloch, Phillip B. Gibbons, and Harsha Vardhan Simhadri. In *Proceedings of the Twenty-Second Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '10, page 189–199, New York, NY, USA, 2010. Association for Computing Machinery.

[7] Guy E. Blelloch, Phillip B. Gibbons, and Harsha Vardhan Simhadri. Low depth cache-oblivious algorithms. In *Proceedings of the Twenty-second Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '10, pages 189–199, New York, NY, USA, 2010. ACM.

[8] Guy E. Blelloch, Charles E. Leiserson, Bruce M Maggs, C Greg Plaxton, Stephen J Smith, and Marco Zagha. An experimental analysis of parallel sorting algorithms. *Theory of Computing Systems*, 31(2):135–167, 1998.

[9] Ernie Chan, Marcel Heimlich, Avi Purkayastha, and Robert van de Geijn. Collective communication: theory, practice, and experience. *Concurrency and Computation: Practice and Experience*, 19(13):1749–1783, 2007.

[10] Zhiwei Chen and Aoqian Zhang. A survey of approximate quantile computation on large-scale data. *IEEE Access*, 8:34585–34597, 2020.

[11] David R. Cheng, Viral B. Shah, John R. Gilbert, and Alan Edelman. A novel parallel sorting algorithm for contemporary architectures, 2007.

[12] Richard Cole. Parallel merge sort. *SIAM Journal on Computing*, 17(4):770–785, 1988.

[13] W Donald Frazer and AC McKellar. Samplesort: A sampling approach to minimal storage tree sorting. *Journal of the ACM (JACM)*, 17(3):496–507, 1970.

[14] Michael T Goodrich. Communication-efficient parallel sorting. *SIAM Journal on Computing*, 29(2):416–432, 1999.

[15] Yan Gu, Julian Shun, Yihan Sun, and Guy E Blelloch. A top-down parallel semisort. In *Proceedings of the 27th ACM symposium on Parallelism in Algorithms and Architectures*, pages 24–34, 2015.

[16] Vipul Harsh, Laxmikant Kale, and Edgar Solomonik. Histogram sort with sampling. In *The 31st ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '19, page 201–212, New York, NY, USA, 2019. Association for Computing Machinery.

[17] Reinhard Heckel, Max Simchowitz, Kannan Ramchandran, and Martin Wainwright. Approximate ranking from pairwise comparisons. In *International Conference on Artificial Intelligence and Statistics*, pages 1057–1066. PMLR, 2018.

[18] Laxmikant V Kale and Sanjeev Krishnan. A comparison based parallel sorting algorithm. In *Proceedings of the International Conference on Parallel Processing (ICPP)*, volume 3, pages 196–200. IEEE, 1993.

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

[20] Tom Leighton. Tight bounds on the complexity of parallel sorting. *IEEE Transactions on Computers*, C-34(4):344–354, 1985.

[21] Xiaobo Li, Paul Lu, Jonathan Schaeffer, John Shillington, Pok Sze Wong, and Hanmao Shi. On the versatility of parallel sorting by regular sampling. *Parallel Computing*, 19(10):1079–1103, October 1993.

[22] Alexander M Mood. The distribution theory of runs. *The Annals of Mathematical Statistics*, 11(4):367–392, 1940.

[23] E. Solomonik and L. V. Kale. Highly scalable parallel sorting. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1–12, April 2010.

[24] Hari Sundar, Dhairya Malhotra, and George Biros. Hyksort: A new variant of hypercube quicksort on distributed memory architectures. In *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing*, ICS '13, pages 293–302, New York, NY, USA, 2013. ACM.

[25] Leslie G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, August 1990.

A APPENDIX: ADDITIONAL PSEUDOCODE AND DIAGRAMS

This appendix provides some supplementary details to our main contributions. Algorithm 1 describes the details of the approach presented in Section 2. Figure 3 provides a pictorial overview of our lower bound argument in Section 3.

Algorithm 1: HSS partitioning algorithm, steps are numbered in the order they are executed

Processors $\mathcal{P}_0, \mathcal{P}_1 ... \mathcal{P}_p$ start with $\mathcal{P}_i$ owning n/p distinct keys contained $\mathcal{A}_i$ with $\mathcal{A} = \bigcup_{i=1}^{p} \mathcal{A}_i$

Initialize bounds to include all elements, e.g., $L_k = I(1), U_k = I(N),\ \forall k \in \{1, \ldots, p-1\}$

Initialize the set of invalid splitters as $W = \{1, \ldots, p-1\}$

while $|W| > 0$ **do**

  At all processors $\mathcal{P}_i$:

  Set the sampling ratio as $\psi = \frac{2p^2}{|W| N \log^* p}$

  Sample each key in $\mathcal{A}_i \cap \left(\bigcup_{k \in W} [L_k, U_k] \right)$ with sampling probability ψ to create subset S_i

  Allgather and merge/sort sample keys to obtain full sample $S = \bigcup_{i=1}^{p-1} S_i$

  Compute local histogram $\mathcal{J}$ for $\mathcal{S}$

  Reduce all local histograms (accumulate $\mathcal{J}$ to obtain $\mathcal{H}$ on $\mathcal{P}_0$)

  At processor $\mathcal{P}_0$:

  Receive combined reduced histogram $\mathcal{H}$ for $\mathcal{S}$ from all $\mathcal{P}_i$

  Update lower/upper bound keys L_k and U_k for all partitions k

  Set s_k to L_k or U_k, depending on whether $R(L_k)$ or $R(U_k)$ is closer to Nk/p for all k

  Update W by removing indices of all splitters that are valid within ϵ error (the k^{th} splitter is valid if $R(s_k) \in [k\frac{N}{p}, k(1+\epsilon)\frac{N}{p}]$)

  Broadcast updates to W and $\{L_k, U_k\}_{k=1}^{p-1}$ to all $\mathcal{P}_i$ for next iteration

  At all processors $\mathcal{P}_i$:

  Receive updates to $\{L_k, U_k\}_{k=1}^{p-1}$ and W from $\mathcal{P}_0$

end

Collect (if necessary) and output splitters $s_1, \ldots s_{p-1}$