An Efficient Parallel Sketch-based Algorithm for Mapping Long Reads to Contigs

Tazin Rahman, Oieswarya Bhowmik, Ananth Kalyanaraman
Washington State University, Pullman, USA

***Abstract*—Long read technologies are continuing to evolve at a rapid pace, with the latest of the high fidelity technologies delivering reads over 10Kbp with high accuracy (99.9%). However, there also exist partially constructed assemblies using short read data. Hybrid assembly workflows provide a way to combine the information in both these data sources and generate highly improved and near complete assemblies and genomic scaffolds. In this paper, we address the problem of mapping long reads to contigs (representing prior constructed partial assemblies). This is a many-to-many comparison application. However, brute force comparisons of all pairs is not practical. Therefore, in this paper, we present a parallel, alignment-free sketching-based algorithm that efficiently maps long reads to contigs. More specifically, our approach uses a minimizer-based Jaccard estimator (or JEM), a variant of the classical MinHashing technique, as its sketch. Experimental evaluation shows that our parallel algorithm is highly effective in producing a high quality mapping while improving significantly the time to solution compared to state-of-the-art mapping tools. For instance, for a large genome *Betta splendens* ($\approx 350Mbp$ genome) with $429K$ HiFi long reads and $98K$ contigs, our JEM approach produces a mapping with 99.31% precision and 96.18% recall, while yielding $7.13\times$ speedup over a state-of-the-art mapper (`Mashmap`).**

***Index Terms*—hybrid assembly, long read mapping, sketching, MinHashing, parallel algorithms**

I. Introduction

Over the last two decades, numerous genomes have been assembled using short read sequencing technologies. These technologies continue to present a cost-effective and high-throughput solution to sequencing. While the short reads are highly accurate ($< 1\%$ error) the challenge is the short read lengths (100 to 250bp)—which also means that the assembled contigs ($\approx 10^3 - 10^4$) tend to be several orders of magnitude short off their genome targets ($\approx 10^6 - 10^9$). Recent emergence in long read sequencing technologies show a promising front toward addressing this challenge. The first generation of long read technologies (e.g., PacBio SMRT [1] or Oxford Nanopore ONT [2]) produce reads that are over 10Kbp but also have a larger error rate (between $11\%-14\%$ [3]). As a significant advancement, the latest generation of technologies such as PacBio HiFi (High Fidelity) [4] have highly improved accuracy (99.9%). There are also several long read error correction tools [5]. Given these, the prospect of assembling long contiguous portions of the genome has dramatically improved [6].

Broadly speaking, two classes of approaches exist for using long reads—standalone and hybrid. *Standalone* long read assemblers [6], [7] produce an assembly directly from long reads; but this also requires a higher sequencing coverage (e.g., $30\times$ or more on most genomes [8]). On the other hand, *hybrid assemblers* [9], [10] offer the benefit of combining long and short reads, or alternatively, combining long reads with prior assembled short read contigs. The power to combine both types of data is particularly attractive if one wishes to build on and extend the reach prior constructed draft assemblies. In addition, use of prior constructed contigs also improves scalability since the number of contigs tends to be orders of magnitude fewer in number compared to raw short reads.

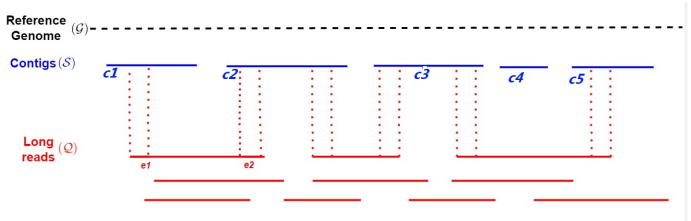

Fig. 1: Example illustrating mapping of long reads to contigs. Each end of a long read can be expected to map to a single contig (assuming a non-redundant contig set).

In this paper, we visit the problem of mapping under hybrid settings—for combining high fidelity long reads, which are gradually becoming mainstay, with contigs assembled from short reads. The motivation for combining long reads with contigs is two fold: a) to help link contigs covering different but nearby parts of the genome (i.e., filling in the assembly gaps); and b) to do so with decreased coverage (and cost) in long read sequencing. In order to implement this hybrid strategy, we need a way to efficiently map the long reads to the contigs, as that step is the primary computational bottleneck in scaffolding. Therefore, we focus on the mapping step for this paper. Fig. 1 shows an example illustration of this mapping process.

While there are a number of sequence mapping solutions (see Section II for a review), scalability of these tools and in addition their ability to work with different types of sequences (short vs. long reads) are limitations.

Contributions: We present a new parallel sketching-based scheme for efficient and scalable mapping of (high fidelity) long reads to contigs (obtained from short reads). The input is a set of long reads (queries $\mathcal{Q}$) and a set of contigs (subjects $\mathcal{S}$). The output is a mapping from $\mathcal{Q}$ to $\mathcal{S}$ such that each long read $r \in \mathcal{Q}$ is mapped to a "best matching" subject $c \in \mathcal{S}$. Key contributions include:

- Methods: We present a new sketching-based method for alignment-free mapping of long reads to contigs. Our approach uses a minimizer-based Jaccard estimator as sketch.
- Implementations: We provide an efficient and scalable parallel distributed memory implementation for our minimizer-based Jaccard estimator workflow.
- Evaluation: We conduct a thorough empirical evaluation of the proposed sketching based implementations. Results show that our method is able to match the mapping quality of a state-of-the-art mapping tool, while providing significant speedups over the state-of-the-art. More specifically, our distributed memory implementation running on 64 processes achieves speedups between $5.6\times$ to $13\times$ compared to the state-of-the-art multithreaded execution on 64 threads.

With mapping applications increasingly becoming more heterogeneous in their data sources, including in hybrid scaffolding/assembly workflows or reference-guided assembly workflows [11], the techniques described in this paper have broad applicability. In what follows, we provide a brief review of the relevant sequence mapping literature (§II), before describing our parallel approach (§III) and presenting the results (§IV).

II. Related work

Sequence mapping is a classical problem in bioinformatics. It can be abstracted as one of mapping a set of *query* sequences (e.g., reads) to a set of *subject* sequences. Traditional sequencing mapping tools (e.g., [12], [13]) focus on aligning short reads (queries) against a reference genome (subject). However, the hybrid setting differs from this classical setting in a couple of different ways. First, *in lieu* of the reference (which is typically a handful of very long subject sequences), our input subjects consist of a set of contigs which represent a highly fragmented view of the reference genome. consequently, the contig sets can have tens to hundreds of thousands of sequences, and may also widely vary in their sequence lengths (10^3-10^5 bp).

As for the query set, long reads are significantly longer than the short reads used in conventional reference mapping. In the absence of more scalable tools, the current batch of hybrid assemblers [9], [10], [14] rely on a classical mapping tool to implement their mapping step. For instance, Haslr [10] first assembles the short reads using Minia [15], and then aligns the contigs to the set of long reads using Minimap2 [13]. Similarly, SAMBA [14] aligns the long reads to the set of contigs using Minimap2 [13].

To improve scalability of mapping, there has been a growing interest in alignment-free approaches [16]–[20], and in particular sketching—e.g., minimizers [20], [21] and MinHashing [22]. Sketching is a class of techniques that use samples derived from the input sets (or sequences) to be compared in order to approximate similarity. Introduced originally for document clustering [22], sketching and its relatives like minimizers [21] have found extensive use in bioinformatics. While these techniques have shown significant promise for mapping in the classical setting, they have not yet been fully explored or evaluated the hybrid use-case targeted in this paper.

III. Methods

Recall that $\mathcal{Q}$ and $\mathcal{S}$ denote the sets of queries and subjects respectively. For the mapping use-case covered in this paper, we set $\mathcal{Q}$ to be the set of input long reads, and $\mathcal{S}$ to the set of input contigs. Intuitively, for each long read in $\mathcal{Q}$ we wish to compute best matching contig(s) in $\mathcal{S}$ (as illustrated in Fig. 1). The rationale for targeting best hit is because a long read query could have emerged from only one place in the genome, and therefore we are after a contig that best covers that unknown region of the target genome. Here, we assume that the set of input contigs are non-redundant, with negligible duplication ratio, which is a reasonable assumption that holds in practice, for most short-read contig assembly tools [15].

Problem statement: Let $m = |\mathcal{Q}|$ and $n = |\mathcal{S}|$. Let $\Sigma = \{\mathsf{a}, \mathsf{c}, \mathsf{g}, \mathsf{t}\}$ denote the DNA alphabet; therefore, $\mathcal{Q} \subseteq \Sigma^*$ and $\mathcal{S} \subseteq \Sigma^*$. Let function `map(q,s)` refer to the process of mapping a query q to a subject s, and a scoring function $\psi : \Sigma^* \times \Sigma^* \rightarrow \mathbb{R}_{\geq 0}$ to denote the quality of the mapping. Consequently, the mapping problem is defined as follows.

Definition 1. Long read-to-contig (`L2C`) mapping: *Given $\mathcal{Q}$ and $\mathcal{S}$, find for each query $q \in \mathcal{Q}$ a subject $s_q^* \in \mathcal{S}$, such that:*

$$s_q^* = \arg\max_s \psi(q, s)$$

Ideally, the function ψ is dictated by a sequence alignment measure. However, computing alignments at scale is very expensive, thereby making alignment-free approaches more desirable in practice. Furthermore, a brute-force approach of comparing every ⟨long read, contig⟩ pair is also not feasible. In fact, in practice we expect a long read to share similarity with only a very small subset of contigs. Therefore, our approach uses an alignment-free sketch to reduce the search space as described below.

A. Preliminaries and notation

a) `MinHash` preliminaries: Since our sketch is based on `MinHash`, we first provide some basic preliminaries about the classical `MinHash` scheme. The `MinHash` sketching scheme was introduced by Broder in 1997 [22], originally to compute resemblance or Jaccard similarity between documents. Given two sets A and B, the Jaccard similarity between the sets is given by: $\mathcal{J}(A, B) = \frac{|A \cap B|}{|A \cup B|}$. In his seminal work, Broder showed that there exists a family of permutations ($\pi : [n] \rightarrow [n]$) called the minwise independent permutations, which can be used to generate fixed size sketches from the sets A and B. It then suffices to compare the sketches instead of explicitly computing the $\mathcal{J}(A, B)$, i.e.,

$$Pr(min\{\pi(A)\} = min\{\pi(B)\}) = \mathcal{J}(A, B)$$

In other words, higher the Jaccard similarity, higher the probability that the sketches obtained A and B will match. To improve the chance that a random sketch is found, a fixed

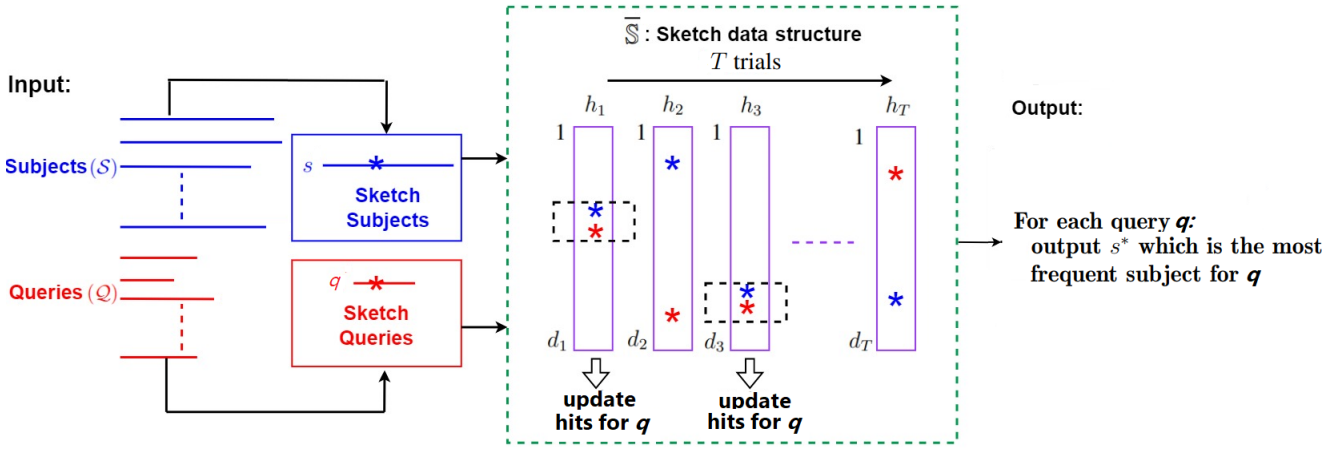

Fig. 2: Illustration of the major steps of our sketch-based algorithm, `JEM-mapper`, for `L2C` mapping.

number of random trials (T) is executed. This is achieved by choosing T random minwise independently permutations: $\{\pi_1, \pi_2, \ldots, \pi_T\}$, and using them to generate the `MinHash` *sketches* for sets A and B, denoted by $\overline{A}$ and $\overline{B}$ respectively:

$$\begin{aligned} \overline{A} &= [\min\{\pi_1(A)\}, \ldots, \min\{\pi_T(A)\}]; \\ \overline{B} &= [\min\{\pi_1(B)\}, \ldots, \min\{\pi_T(B)\}] \end{aligned}$$

Subsequently, if any of the trials produce the same minimum between $\overline{A}$ and $\overline{B}$ then we conclude A is *similar* to B. In practice, a value around 100 to 200 is used for T [22]. We refer to the `MinHash` sketches (e.g., $\overline{A}$, $\overline{B}$) sometimes as just "MinHashes" for the underlying sets.

b) String notation: Let s denote an arbitrary string over alphabet Σ, and let $|s|$ denote its length. We use the terms strings and sequences interchangeably. A k-mer is a (sub)string of length k. Given Σ and k, let $\mathcal{K}$ denote the set of all k-mers that can be built using Σ. Note that $|\mathcal{K}|=|\Sigma|^k$. We use the term *canonical ordering* of k-mers, denoted by Π_k^*, to refer to the lexicographical ordering of the k-mers in $\mathcal{K}$. For instance, if k=2, the canonical ordering of $\mathcal{K}$ is given by: Π_k^* = [aa,ac,ag,at,ca,cc,cg,ct,ga,gc,gg,gt,ta,tc,tg,tt]. Given a string $s \in \Sigma^*$ and a choice of k, the notation s_k is used to denote the set of all k-mers present in s.

B. Computing `L2C` using a minimizer-based Jaccard estimator sketch

In the `L2C` mapping application, we have two sets of strings—$\mathcal{Q}$ containing long reads, and $\mathcal{S}$ containing contigs. For a string s, its `MinHash` sketch can be constructed from the set of all k-mers (s_k) in s—i.e., during trial t, apply a hash function $h_t(.)$ on each k-mer in s_k and then select the k-mer with the minimum value as part of the sketch.

Using this idea, a simple way to apply the MinHashing scheme for `L2C` is as follows. First enumerate the MinHashes (or the sketches) for each subject (one minimum for each random trial $t \in [1, T]$) and insert those into a sketch data structure $\overline{\mathbb{S}}$. Subsequently, during querying time, sketches are also generated from each query. The more sketches a query generates in common with a subject, the higher the likelihood that it shares a high sequence level similarity. Therefore, we can simply track the frequency of the subjects that "hit" with a given query, and report the top matching subject (if any) as the mapped output hit to that query. This simple algorithmic workflow is illustrated in Fig. 2.

While this workflow can be efficiently implemented, we make several changes, as there are a few key challenges with a direct application of MinHashing as described above. First, note that in the `L2C` application, contigs and long reads could have significantly differing lengths. If a long read r is significantly longer (say $10Kbp$) than a corresponding mapped contig c (say $3Kbp$), then even if c has significant identity within r, MinHashing may select k-mers that may lie outside the region of the overlap. This could mean missing out on a true mapped (affects recall). Similarly, if a contig c is significantly longer (say, $20Kbp$) compared to a long read r (say $10Kbp$), recall could again be affected as the sketches from contigs could lie outside the region of true overlap. Either way, the qualitative efficacy of `MinHash` for mapping could be negatively impacted.

To overcome this limitation, we use two ideas: a) to map only end segments of long reads; and b) to compute a minimizer-based Jaccard estimator (instead of the classical `MinHash` form).

1) Using the ending segments of a long read: Instead of extracting sketches from the entire length of a long read, our approach uses only the ending regions (aka. "end segments") of a long read. Specifically, we define a fixed *segment length* ℓ. We then map only the first ℓ characters (prefix segment) and the last ℓ characters (suffix segment) of a long read, and reporting their respective best hit contigs. This approach has two advantages: a) It provides the subsequent hybrid scaffolding step information about the farthest separated pair of contigs that are linked by this long read (increasing the span of the scaffold over the target genome); and b) The approach avoids the problem of generating sketches from interior regions of the long read, are not important in scaffolding applications. This

not only improves quality, but also reduces work, making the algorithm faster. (Note that for non-scaffolding applications, this segment-based approach may not apply to cases where a contig may be completely contained within an interior region of a long read. In such cases, an extension of the approach will be needed.) Henceforth, we revise the set of queries $\mathcal{Q}$ to include the prefix and suffix segments of each long read—i.e., if there are m long reads, then $\mathcal{Q}$ consists of $2m$ sequences of length ℓ each. We used a value of $\ell = 1000$ in all our experiments. Fig. 1 shows the end segments of long reads mapped to contigs.

2) Sketching using minimizer Jaccard estimate: The end segment idea constrains the regions where sketches are extracted from the long reads. However, the contigs (subjects) can still be very long, and while an end segment of a long read is expected to span only a short region ($\approx \ell$ bp) along the contig, the location of that mapped region is unknown *a priori*. Therefore, we follow a two-pronged idea: a) reducing the base set of k-mers for Jaccard similarity computation to a set of *minimizers* [21] obtained from contigs, and b) then using a sliding interval of length ℓ bp over the list of those minimizers to select one `MinHash` per interval. The list of minhashes so extracted becomes our version of the minimizer-based Jaccard estimator sketch (abbreviated as "JEM" henceforth) of the subject for that trial $t \in [1, T]$. Fig. 3 illustrates this procedure using a conceptual example. The detailed algorithm is as follows.

The minimizer-based Jaccard estimate calculates the Jaccard similarity between two sequences using the minimizer sketches between them. Let us consider a uniformly random hash function. Given a sequence s, an integer k and window size w, the minimizer is the k-mer with smallest hash value of the w consecutive k-mers. We use the lexicographically smallest k-mer as this hash function, consistent with previous works [23], [24]. The *minimizer sketch* of s (denoted by $M(s, w)$) is the set of all minimizers in s. Hence the minimizer Jaccard estimate between sequences A and B is:

$$\mathcal{J}_m(A, B; w) = \mathcal{J}(M(A, w), M(B, w))$$

In other words, the minimizer Jaccard estimate allows us to collect and compare sketches from the list of minimizers of a sequence (rather than all the k-mers). This reduces work and also provides a certain degree of qualitative robustness against noisy k-mers.

Minimizer Jaccard estimate has been used prior in the widely used mapping tool Mashmap [16], [25]. Our algorithm is different in the way these sketches are computed. More specifically, in Mashmap, for each minimizer, a list of all positions it is present in the subject is maintained. Later, during mapping time, if a query shares a minimizer with multiple positions, then the region where the query has maximal local intersection on the subject is detected and reported at query time. This approach entails one of short listing positional candidates and then eliminating those that do not have sufficient concentration of query minimizers in their vicinity.

By contrast, our approach directly applies the end segment length ℓ of the long read query as the interval length over the subject, and tracks the `MinHash` for each such interval of the subject—as shown in Fig. 3. This guarantees that the sketches are generated at the resolution of the end segment length, both for the subjects and queries, thereby obviating the need to check for any distance constraints later.

Algorithm 1: $Sketch\_byJEM$

Input: s: input sequence
ℓ: segment length
$\mathcal{H}$: set of T hash functions $\{h_1, h_2, ..., h_T\}$
Output: sketches generated by for s

1: $Sketch \leftarrow []$
2: Let $s_k \leftarrow$ the set of all k-mers in s
3: $M_o(s, w) \leftarrow Generate\_Minimizers(s_k, w)$
`/* returns a list of minimizer tuples` $\langle k_i, p_i \rangle$`, sorted by position index` p_i `*/`
4: **for** each tuple $\langle k_i, p_i \rangle \in M_o(s, w)$ **do**
5: $M_i \leftarrow Generate\_Interval(m_k, i, \ell)$ `/* returns the set of minimizers` $\{\langle k_j, p_j \rangle : p_i \leq p_j \leq p_{i+\ell}\}$ `*/`
6: **for** each trial $t \in [1, T]$ **do**
7: $sketch \leftarrow \arg\min_{x \in M_i} h_t(x)$
8: $Sketch[t]$.insert(key: $sketch$, value: s)
9: **end for**
10: **end for**
11: **return** $Sketch$

More specifically, let $M_o(s, w)$ represent the set of all minimizer tuples $\langle k_i, p_i \rangle$ from subject s, where k_i denotes the minimizer at position p_i on s. The set $M_o(s, w)$ is kept sorted based on the minimizer positions. For a given interval length ℓ, let us define M_i to be the set of consecutive minimizers in $M_o(s, w)$ such that $M_i = \{\langle k_j, p_j \rangle : p_i \leq p_j \leq p_i + \ell\}$. We slide intervals of length ℓ over the set of minimizers and within each interval, T minhashes are generated. Algorithm 1 shows how sketches are extracted using our approach. Algorithm 2 shows the overall `JEM-mapper` algorithm.

Implementation notes: In our implementation, we have used lexicographic ordering of k-mers to extract minimizers from window w. For a substring $s^{'}$ of length greater than k, a canocical minimizer is the smallest k-mer of $s^{'}$ and its reverse complement $\overline{s}^{'}$ based on lexicographic ordering. To generate the T minhashes for each interval, we assign each minimizer of that interval its k-mer rank x (i.e., as per its canonical ordering in $\mathcal{K}$), and then use T random hash functions of the linear congruential form: $h_t(x) = (A_t \cdot x + B_t) \mod P_t$. Subsequently, the k-mer corresponding to the smallest hashed value becomes the sketch for that string for trial t. Here A_t, B_t, and P_t are randomly generated constants (and generated *a priori*).

C. Parallelization

We present a distributed memory parallel algorithm for Algorithm 2, with code written in C/C++ and

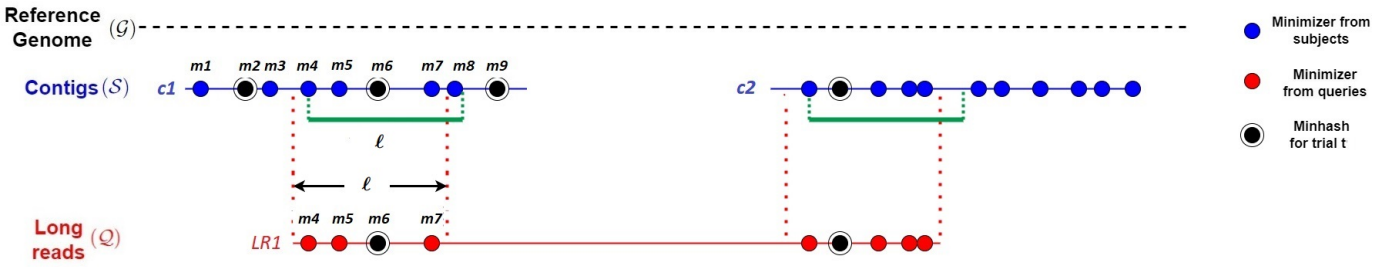

Fig. 3: An example to illustrate the way minimizer-based Jaccard estimate sketch (or the JEM sketch) is generated by our approach, `JEM-mapper`. At the subject processing time, the list of minimizer tuples $M_o(s, w)$ is generated for each contig (shown as red circles). We then slide an interval of length ℓ over the set of minimizers based on their positions. On $c1$, this is shown as the list $[m_4 \dots m_8]$. For each such interval, we generate T minhashes for T trials. The black concentric circle shows the minhash that was randomly selected for trial t in that interval (i.e., the sketch). At query processing time, for each end of the long read, we generate a similar set of minimizers (denoted by the red circles). We then pick T JEM sketches in a similar fashion and look for hits in the sketch table before detecting the top hit.

Algorithm 2: `L2C` mapping by `JEM-mapper` $(\mathcal{Q}, \mathcal{S})$

Input: $\mathcal{Q}$: long read segments, $\mathcal{S}$: contigs, T: no. trials
Output: $\Phi : \mathcal{Q} \rightarrow \mathcal{S}$

1: Initialize sketch table: $\overline{\mathbb{S}}[t] \leftarrow \phi$, where $t \in [1, T]$
2: $\overline{\mathbb{S}}$.insert(Sketch_byJEM(c)), $\forall c \in \mathcal{S}$
3: **for** each $r \in \mathcal{Q}$ **do**
4: $\quad\overline{\mathbb{S}}$.lookup(Sketch_byJEM($r$))
5: $\quad$**for** $t \in [1, T]$ **do**
6: $\quad\quad$Let $Hits_r[t] \leftarrow \{c | r$ and c collide in $\overline{\mathbb{S}}[t]\}$
7: $\quad$**end for**
8: $\quad\Phi(r) = c^*$, where c^* is the most frequent contig in $Hits_r$
9: **end for**
10: **return** Φ

MPI for communication. The beta version of this software is available on https://github.com/TazinRahman1105050/JEM-Mapper/tree/beta-2.1. We use the following notation: $m = |\mathcal{Q}|$; $M = \Sigma_{r \in \mathcal{Q}} |r|$; $n = |\mathcal{S}|$; $N = \Sigma_{c \in \mathcal{S}} |c|$; and p to denote the number of processes. The major parallel steps are as follows.

S1) *(load input)* The processes load the input $\mathcal{Q}$ and $\mathcal{S}$ in a distributed manner, such that each process gets approximately $\mathcal{O}(\frac{M}{p})$ query bases and $\mathcal{O}(\frac{N}{p})$ subject bases. Let $\mathcal{Q}_{local}$ and $\mathcal{S}_{local}$ denote the sets of local queries and subjects respectively, held by any given process.

S2) *(sketch subjects)* Each process generates the sketches from $\mathcal{S}_{local}$, and inserts them into $\overline{\mathbb{S}}_{local}$, which holds all the sketches generated from that process.

S3) *(gather sketch)* In a global communication step that uses the `MPI_Allgatherv` primitive, we perform a union of all the $\overline{\mathbb{S}}_{local}$ into a single $\overline{\mathbb{S}}_{global}$ that is stored at each process. Note that each $\overline{\mathbb{S}}_{global}$ consists of T lists, one for each trial, as shown in Fig. 2.

S4) *(map queries)* Each process then processes its local query set $\mathcal{Q}_{local}$. The mapping step for each long read $r \in \mathcal{Q}_{local}$ comprises of three steps: a) sliding window and generate its JEM sketches, b) lookup the contig hits in $\overline{\mathbb{S}}_{global}$, and c) report mapping for the (or a) best hit. As shown in Fig. 2, hits are located within $\overline{\mathbb{S}}_{global}$ by the corresponding trial numbers (step b). Subsequently, a reporting step scans the bins (or the list of trials) to generate the mapping output pairing queries to subjects (step c).

Implementation notes: For step (c) above, we implemented a lazy update strategy to support a fast tracking of subjects and their hit rates, *across queries*. More specifically, we initialize an array $A[1, n]$ of tuples of the form $\langle u, v \rangle$, where u is an integer counter initialized to 0, and v is the query id (initialized to -1). Whenever a query j generates a hit with a subject i, we check if $A[i].v$ is equal to j. If it is, then we simply increment counter $A[i].u$. But if it is not, then we first set $A[i].v$ to j, reset counter $A[i].u$ to 0, and then increment that counter (to 1). This lazy strategy avoids the cost of resetting the counters for all subjects whenever a new query is processed. Note that at each process, queries in $\mathcal{Q}_{local}$ are processed one by one.

1) Complexity analysis: The input loading step (S1) costs $\mathcal{O}(\frac{N+M}{p})$ time. Sketching the subjects (S2) can be achieved in $\mathcal{O}(\frac{n\ell_s T}{p})$ time, where ℓ_s is the average length of a subject. Similarly, sketching the queries (S4) can be achieved in $\mathcal{O}(\frac{m\ell_q T}{p})$ time, where ℓ_q is the average length of a query. The gather step (S3) involves communicating each $\overline{\mathbb{S}}_{local}$ to all processes, and can be achieved in $\mathcal{O}(\tau \log p + \mu n T)$ time, where τ is the cost of network latency and μ is the reciprocal of network bandwidth (i.e., sec per byte transferred). While the worst-case size of $\overline{\mathbb{S}}_{global}$ is $\mathcal{O}(n\ell_s T)$, in practice we expect significantly fewer minhashes because we are selecting from the list of minimizers $M_o(s, w)$ (and not all k-mers). Finally, the query mapping step (S4) is a local step processing each query $r \in \mathcal{Q}_{local}$. The initialization of counters for the subjects

takes $\mathcal{O}(n)$ time and after that each query is mapped through a linear scan of its sequence (with T minhash computations at all its minimizers). Consequently, step S4 takes $\mathcal{O}(n + \frac{m\ell_q T}{p})$ time. Since the number of long reads (m) can be expected to be significantly more than the number of contigs (n) due to sequencing coverage, we expect $\frac{m\ell_q T}{p}$ to dominate over n in practice.

The space complexity of our approach is dominated by the size of $\overline{\mathbb{S}}_{global}$. Let m_s denote the average number of minimizers generated per subject. Since we enumerate fixed-size intervals and store one minhash per interval, a subject s can be expected to contribute up to $\mathcal{O}(m_s T)$ minhashes into the sketch. Therefore, the space complexity per process is $\mathcal{O}(nm_s T)$.

IV. Experimental Results

In this section, we present a thorough experimental evaluation of our sketch-based mapping algorithm, `JEM-mapper` (§ III-B)). We study both quality and performance, using both simulated and real-world data sets.

A. Experimental setup

a) Test inputs: We used two sets of long read inputs ($\mathcal{Q}$) in our experiments (see Table I):

- *PacBio HiFi sim. long reads:* generated using the Sim-it PacBio HiFi read simulator [26], run with a low coverage of 10× and a long read median length 10Kbp; and
- *PacBio HiFi real long reads:* real-world PacBio HiFi reads for *Oryza sativa* (chr 8) downloaded from the PacBio repository [27].

The first simulated read data set allows us to evaluate using a ground-truth (using the coordinate information acquired using any existing mapping tool), while the real-world data set is aimed at a real-world application. Simulated read inputs were generated from real-world genomes, downloaded from NCBI GenBank [28], for six different organisms ranging from bacterial to eukaryotic (listed in Table I). The organisms are *E. coli*, *P. aeruginosa*, *C. elegans*, *D. busckii*, Human-7 and 8 chromosome, and *B. splendens*. Once the long reads are generated, we pulled out the two end segments (prefix and suffix) of length $\ell = 1,000$ bp and added them to the respective query set $\mathcal{Q}$.

We used the following two steps to construct the contigs ($\mathcal{S}$) for all the inputs: use the ART sequencing simulator [29] to generate 100bp Illumina short reads; and assemble the short reads using the Minia assembler [15] into contigs ($\mathcal{S}$).

b) Test platform: All experiments were conducted on a distributed memory cluster with 9 compute nodes, each with 64 AMD Opteron™ (2.3GHz) cores and 128 GB DRAM. The nodes are interconnected using 10Gbps Ethernet and share 190TB of ZFS storage. The cluster supports OpenMPI and OpenMP.

c) Software configuration: All runs of our software `JEM-mapper` was performed using the following set of parameters as its default: k-mer size $k = 16$; no. trials $T = 30$ (choice explained in Fig. 6); and window size to generate minimizer sketches $w = 100$. In other words, we select a k-mer (of size 16 bp) from a consecutive stretch of w (100) k-mers to be the minimizer. Minimizers are added to the corresponding set $M_o(s, w)$ only if they change or if the current minimizer goes out of bounds. Subsequently, to generate the JEM sketches (Algorithm 1), we set the interval length same as the end segment ℓ bp (or 1,000 bp) for long reads.

For comparative evaluation, we compared against two state-of-the-art reference genome mappers, namely `Mashmap` [16] and `Minimap2` [13]. Of these two, `Mashmap` tool [16] is a fast reference genome mapper that also uses sketching, and from its implementation we can easily extract the top hit for a query, making it possible to do a head to head comparison with `JEM-mapper`. As for `Minimap2` [13], it follows a more classical seed and extend, alignment-based approach, but it also benefits from the use of minimizers internally for the seeding step. However, it was not possible to make a direct comparison with its output because it reports multiple hits for each query. Therefore, we focus our comparative evaluation on `Mashmap`. In all cases, the *same* inputs ($\mathcal{Q}$,$\mathcal{S}$) were provided to both programs—i.e., mapping the end segments of long reads to contigs.

B. Evaluation Methodology

For quality evaluation, we constructed a *benchmark* for all simulated data sets using the coordinate information of the contigs ($\mathcal{S}$) and long reads ($\mathcal{Q}$) mapped back to the full-length reference genome ($\mathcal{G}$). This is illustrated in Fig. 4. More specifically, to determine the ⟨start,end⟩ coordinates of each contig, we mapped the set of contigs to the reference $\mathcal{G}$ using Minimap2 [13]. In the same way, we extracted the coordinates of the long reads. A given end segment of a long read $e \in \mathcal{Q}$ is said to *map* to a contig $c \in \mathcal{S}$ if and only if its respective ⟨start,end⟩ coordinates intersect in at least k positions of the reference genome, where k is the k-mer size—as shown in Fig. 4.

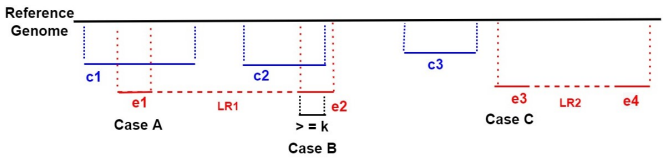

Fig. 4: Benchmark cases for when an ending segment of a long read successfully maps (Cases A and B) or does not map (Case C) to a contig. In the figure, two long reads are shown, one with ends $(e1, e2)$ and another with ends $(e3, e4)$.

Let Bench denote the set of all true ⟨read end,contig⟩ mappings. Let Test denote the set of output ⟨read end,contig⟩ mappings produced by one of our implementations. Then, we classify each ⟨read end e,contig c⟩ pair as:

- **True Positive (TP):** if $\langle e, c\rangle \in$ Test and $\langle e, c\rangle \in$ Bench
- **False Positive (FP):** if $\langle e, c\rangle \in$ Test and $\langle e, c\rangle \notin$ Bench
- **False Negative (FN):** if $\langle e, c\rangle \notin$ Test and $\langle e, c\rangle \in$ Bench
- **True Negative (TN):** if $\langle e, c\rangle \notin$ Test and $\langle e, c\rangle \notin$ Bench

Input genome		$\mathcal{S}$: Subject statistics (Minia contigs)			$\mathcal{Q}$: Query statistics (HiFi sim. long reads)		
Input	Genome length (in bp)	No. contigs ($\geq$ 500bp)	Total subject size in bp (M)	Contig length (avg.±std.dev)	No. long reads (n)	Total query size in bp (N)	Read length (avg.±std.dev)
E. coli	4,641,652	365	4,521,741	12,388 ± 13,997	4,541	46,305,093	10,205 ± 3,418
P. aeruginosa	6,264,404	460	6,155,889	13,382 ± 18,218	6,122	62,504,041	10,221 ± 3,363
C. elegans	100,286,401	30,883	85,664,920	2,819 ± 4,663	98,103	1,001,061,602	10,205 ± 3,400
D. busckii	118,492,362	43,006	109,278,105	2,541 ± 3,151	123,781	1,258,889,285	10,168 ± 3,412
Human chr 7	159,345,973	55,331	111,086,154	2,007 ± 1,934	156,357	1,593,462,533	9,612 ± 2,988
Human chr 8	145,138,636	53,821	110,539,506	2,053 ± 1,876	142,102	1,449,205,836	10,200 ± 3,402
B. splendens	339,050,970	98,160	339,804,114	3,462 ± 4,181	429,520	4,371,221,619	10,177 ± 3,403
O. sativa chr 8 (real)	28,443,022	9,945	18,416,389	1,851 ± 2,067	532,667	10,458,872,536	19,642 ± 4,246

TABLE I: Input data sets used in our experiments.

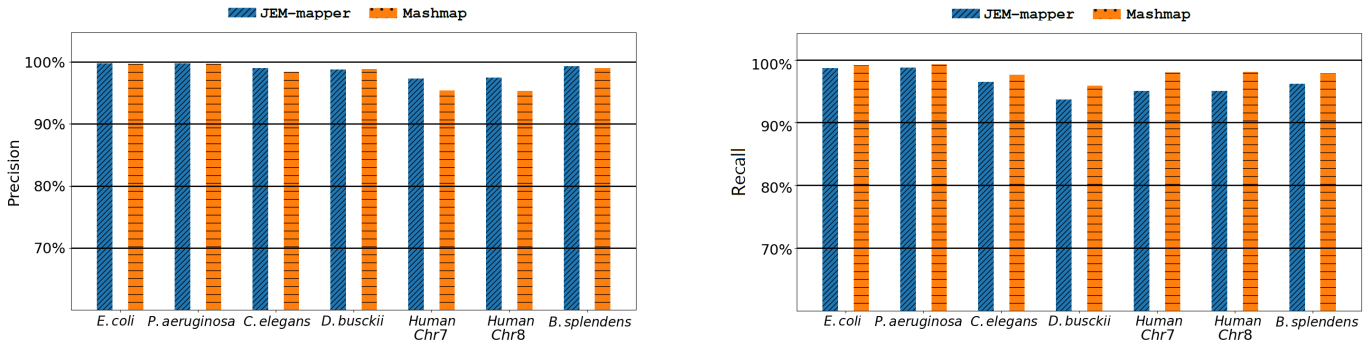

Fig. 5: Quality results (precision and recall) using PacBio HiFi sim. long reads for the different sketch-based schemes.

Based on the above four measures, we calculate precision as $\frac{TP}{TP+FP}$ and recall as $\frac{TP}{TP+FN}$. Note that if an output mapping is a false positive, then by implication it is also a false negative (since there is room for only one best hit). But there can be additional false negatives. Therefore the recall values are upper bounded by the precision values in this evaluation scheme.

C. Qualitative Evaluation

Fig. 5 shows the precision (left) and recall (right) values for `JEM-mapper` and `Mashmap` (for state-of-the-art comparison). These results are for the PacBio HiFi sim. long reads. The results show that by and large, our sketch-based implementation is competitive and show comparable quality compared to `Mashmap` in all cases, with both tools producing well over 95% precision for all inputs tested. For the smaller/less complex genomes (*E. coli*, *P. aeruginosa*) `JEM-mapper` produces similar precision values as `Mashmap`. Our scheme produces better precision for all the larger (more complex) inputs. Eukaryotic inputs have more repetitive content that may lead to reduced precision and the results show that the strategy to select the best candidate from multiple random trials makes our sketch-based scheme more precise for these more complex inputs.

For all the input datasets, `Mashmap` produces better recall as compared to `JEM-mapper`. However, the difference between the two tools is marginal in all cases. Again, both tools produce recall values that are 95% or more for most inputs. We also note that the recall values are very close to the precision values, implying that most of the loss in recall can be attributed to false positive mapping in the top hit. Note that if we are to extend our method to report a fixed number, say top x hits per read, then several of the missing contig hits could possibly be recovered.

The number of trials T could have an impact on the overall quality. Fig. 6 shows the effect of varying T on the `JEM-mapper` and classical `MinHash` implementation, using the *B. splendens* input. Increasing T improves both precision and recall. This behavior is consistent with the fact that with more trials, the sketch-based schemes get more chances to find a hit between a long read and a contig, thus making the recall better. Since these are 99.9% correct long reads, precision also improves. We can observe that `JEM-mapper` can achieve above 95% precision and recall only using 20 trials. After 30 trials it reaches to saturation for precision and recall values and adding more trials only improves precision and recall marginally. However, for classical `MinHash`, even after using 100 iterations, the precision and recall values are quite low compared to `JEM-mapper`. This behavior is expected as `JEM-mapper` is better equipped to identify the region within ℓ-long segment stretches of the subject. In contrast, classical `MinHash` does not constrain identification of sketches from within such distance bounds, and therefore, may need more random trials to recover the hits. The property of supporting fewer number of trials also provides a significant advantage to `JEM-mapper` toward faster performance. For example, to achieve roughly the same quality in mapping on the *B. splendens* input, `JEM-mapper` took 30 trials, whereas the classical `MinHash` implementation took 150 trials.

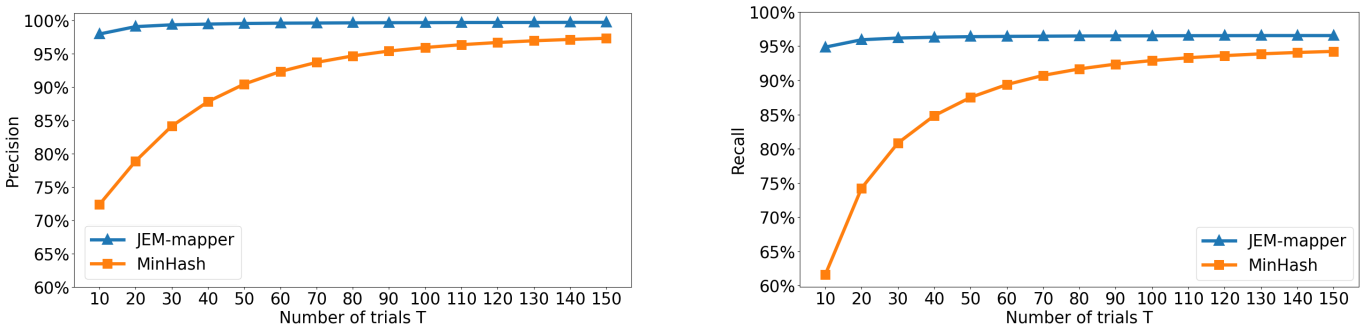

Fig. 6: Effect of varying the number of trials on quality results (precision and recall), using input *B. splendens*. We can observe that `JEM-mapper` can achieve above 95% precision and recall only using 20 trials whereas classical `MinHash` needed to use more than 150 trials to reach a similar quality output.

D. Performance Evaluation

We studied the strong scaling behavior of our parallel implementation for `JEM-mapper`, by varying p from 4 through 64. Table II shows the parallel runtimes for the larger inputs tested. Overall, the parallel runtime reduces with increase in p, demonstrating improving speedups. For instance, on *B. splendens*, the relative speedup (relative to $p = 4$) increases from 1.81× on $p = 8$, to 2.89× on $p = 16$, 3.52× on $p = 32$, and 4.11× on $p = 64$. As the number of processes increase, the work per process also reduces leading to parallel overheads slowly starting to dominate. We have compared our distributed memory implementation results with `Mashmap` runtimes. `Mashmap` supports a multithreaded shared memory implementation. Table II shows the `Mashmap` runtimes for where the number of threads is set to 64. The results show that `JEM-mapper` is significantly faster than the `Mashmap` implementations. In all the input cases, `JEM-mapper` running in distributed memory mode with $p = 64$ yields higher speedup (ranging from 5.6× to 13×) over `MashMap` running on the same number of processors (no. threads = 64).

Fig. 7a (left) shows the parallel runtime broken down by the individual steps of the `JEM-mapper` implementation for $p = 16$. It is evident that the dominant step is the query processing time, which includes the time to sketch the queries and search in the hash table and report the hits.

We also closely analyzed the query processing time from the perspective of querying throughput, defined as the number of queries processed per unit time (sec). To calculate this, we included the times for sliding windows on the queries, sketching the queries, and search in $\overline{\mathbb{S}}_{global}$ and report step. Fig. 7b (right) shows the querying throughput for our `JEM-mapper` implementation, for the larger inputs tested. We observe that this querying throughput scales almost linearly. Notably, these throughputs do not vary with the inputs or their sizes (except for *O. sativa chr 8 (real)*), suggesting high parallel efficiency for this dominant step of the algorithm.

Fig. 8 shows the total computation versus communication time for *Human chr 7* and *B. splendens* varying the number of processors from $p = 4$ to $p = 64$. The total computation time includes the I/O time, subject processing time, generating the $\overline{\mathbb{S}}_{global}$ time, and the query processing and search time. As expected, increasing the number of processors increases the total communication overhead, but the overhead stays well under 25% for up to $p = 64$.

E. Evaluation on real world data set

As a real-world application, we used a real-world PacBio HiFi long read data set for *Oryza sativa* (rice MH63), downloaded from the PacBio repository [27] (see Table I for input statistics). We used only the long reads from chromosome 8 for our experimental analysis ($n = 532K$). We then used `JEM-mapper` to map these long reads to the contigs generated using a Minia assembly of *O. sativa* chr. 8 short reads ($m = 9.9K$). Finally, we used BLAST [30] to compute the percent identity between each long read (segment) and the corresponding mapped contig as reported by `JEM-mapper`. Fig. 9 shows that the percent identity between most of the long read ends and the corresponding contig falls between 95%-100%—showing the high quality of mapping generated by `JEM-mapper`.

V. Conclusions

Hybrid workflows that combine long reads with short-read assemblies have a key role to play as sequencing technologies continue to evolve in accuracy and read length. In this paper, we presented a minimizer-based Jaccard estimator sketch-based algorithm for mapping long reads to contigs. When applied and tested for mapping long reads to short read assembled contigs, our approach produces mapping of high quality while delivering significant speedups over state-of-the-art mapping solutions. Our study also shows the clear benefits of using a minimizer-based Jaccard estimator sketch over the classical MinHash for this application.

This work has opened up multiple avenues for future research, including (but not limited to): i) algorithmic optimizations to further improve quality of mapping while enhancing scaling; ii) several extensions including end-to-end hybrid

Input	JEM-mapper					Mashmap
	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$t = 64$
C. elegans	121	66	41	35	32	201
D. busckii	150	83	51	42	39	219
Human chr 7	187	103	63	48	48	624
Human chr 8	173	96	59	45	44	467
B. splendens	518	285	179	147	126	899
O. sativa chr 8 (real)	420	218	122	91	69	605

TABLE II: Strong scaling results for `JEM-mapper`: Shown are the parallel runtimes (in sec) for `JEM-mapper` as function of the number of processes (p) on various inputs. Also shown is `Mashmap` runtimes. This tool supports only multithreaded shared memory parallelism. We ran it with the number of threads set to 64.

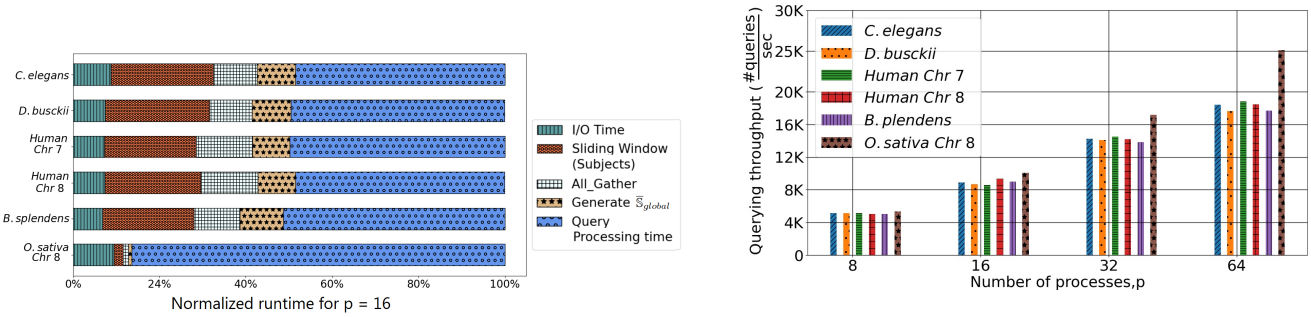

Fig. 7: (a) Runtime breakdown by steps for `JEM-mapper`; (b) Querying throughput for `JEM-mapper`.

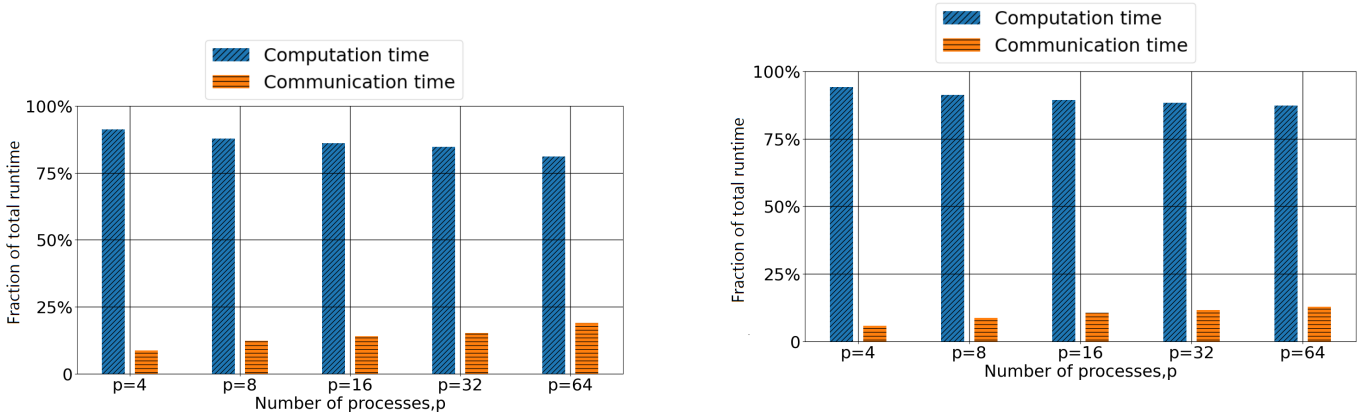

Fig. 8: Fraction of computation time and communication time (in percentages) for (a) *Human chr 7*; and (b) *B. splendens*;

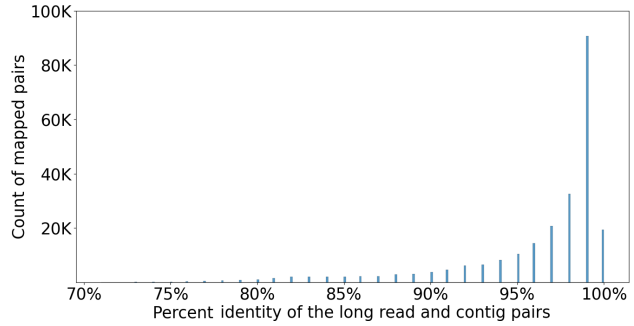

Fig. 9: Percent identity distribution for long read mappings generated by `JEM-mapper` on the *O. sativa* data set.

assembly and scaffolding, and extension to other types of hybrid settings; and iii) large-scale studies targeting more complex eurakyotic genomes. iv) in reference-guided assembly pipelines [11] either reads are mapped against the reference genome or alternatively contigs or scaffolds are aligned against the reference genome. These use-cases can easily benefit from the efficient sketch-based algorithmic template for mapping sequences of varied lengths.

ACKNOWLEDGMENTS

This research was supported in parts by NSF grants OAC 1910213 and CCF 1919122. We thank Dr. Priyanka Ghosh for several discussions during the early stages of the project.

REFERENCES

[1] C. E. Mason and O. Elemento, "Faster sequencers, larger datasets, new challenges," 2012.

[2] D. Deamer, M. Akeson, and D. Branton, "Three decades of nanopore sequencing," *Nature biotechnology*, vol. 34, no. 5, pp. 518–524, 2016.

[3] M. O. Pollard, D. Gurdasani, A. J. Mentzer, T. Porter, and M. S. Sandhu, "Long reads: their purpose and place," *Human molecular genetics*, vol. 27, no. R2, pp. R234–R241, 2018.

[4] T. Hon, K. Mars, G. Young, Y.-C. Tsai, J. W. Karalius, J. M. Landolin, N. Maurer, D. Kudrna, M. A. Hardigan, C. C. Steiner, *et al.*, "Highly accurate long-read hifi sequencing data for five complex genomes," *Scientific data*, vol. 7, no. 1, pp. 1–11, 2020.

[5] P. Morisse, T. LeCroq, and A. LeFeBVre, "Long-read error correction: a survey and qualitative comparison," *BioRxiv*, pp. 2020–03, 2021.

[6] S. Koren, B. P. Walenz, K. Berlin, J. R. Miller, N. H. Bergman, and A. M. Phillippy, "Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation," *Genome research*, vol. 27, no. 5, pp. 722–736, 2017.

[7] G. M. Kamath, I. Shomorony, F. Xia, T. A. Courtade, and N. T. David, "HINGE: long-read assembly achieves optimal repeat resolution," *Genome research*, vol. 27, no. 5, pp. 747–756, 2017.

[8] M. Kolmogorov, J. Yuan, Y. Lin, and P. A. Pevzner, "Assembly of long, error-prone reads using repeat graphs," *Nature biotechnology*, vol. 37, no. 5, pp. 540–546, 2019.

[9] D. Antipov, A. Korobeynikov, J. S. McLean, and P. A. Pevzner, "hybridSPAdes: an algorithm for hybrid assembly of short and long reads," *Bioinformatics*, vol. 32, no. 7, pp. 1009–1015, 2016.

[10] E. Haghshenas, H. Asghari, J. Stoye, C. Chauve, and F. Hach, "Haslr: Fast hybrid assembly of long reads," *Iscience*, vol. 23, no. 8, p. 101389, 2020.

[11] H. E. Lischer and K. K. Shimizu, "Reference-guided de novo assembly approach improves genome reconstruction for related species," *BMC bioinformatics*, vol. 18, no. 1, pp. 1–12, 2017.

[12] B. Langmead, C. Trapnell, M. Pop, and S. L. Salzberg, "Ultrafast and memory-efficient alignment of short dna sequences to the human genome," *Genome biology*, vol. 10, no. 3, pp. 1–10, 2009.

[13] H. Li, "Minimap2: pairwise alignment for nucleotide sequences," *Bioinformatics*, vol. 34, no. 18, pp. 3094–3100, 2018.

[14] A. V. Zimin and S. L. Salzberg, "The SAMBA tool uses long reads to improve the contiguity of genome assemblies," *PLoS computational biology*, vol. 18, no. 2, p. e1009860, 2022.

[15] R. Chikhi and G. Rizk, "Space-efficient and exact de bruijn graph representation based on a bloom filter," *Algorithms for Molecular Biology*, vol. 8, no. 1, pp. 1–9, 2013.

[16] C. Jain, A. Dilthey, S. Koren, S. Aluru, and A. M. Phillippy, "A fast approximate algorithm for mapping long reads to large reference databases," in *International Conference on Research in Computational Molecular Biology*, pp. 66–81, Springer, 2017.

[17] B. D. Ondov, T. J. Treangen, P. Melsted, A. B. Mallonee, N. H. Bergman, S. Koren, and A. M. Phillippy, "Mash: fast genome and metagenome distance estimation using minhash," *Genome biology*, vol. 17, no. 1, pp. 1–14, 2016.

[18] B. D. Ondov, G. J. Starrett, A. Sappington, A. Kostic, S. Koren, C. B. Buck, and A. M. Phillippy, "Mash screen: high-throughput sequence containment estimation for genome discovery," *Genome biology*, vol. 20, no. 1, pp. 1–13, 2019.

[19] G. Marçais, D. DeBlasio, P. Pandey, and C. Kingsford, "Locality-sensitive hashing for the edit distance," *Bioinformatics*, vol. 35, no. 14, pp. i127–i135, 2019.

[20] L. Coombe, J. X. Li, T. Lo, J. Wong, V. Nikolic, R. L. Warren, and I. Birol, "LongStitch: High-quality genome assembly correction and scaffolding using long reads," *BMC bioinformatics*, vol. 22, pp. 1–13, 2021.

[21] M. Roberts, W. Hayes, B. R. Hunt, S. M. Mount, and J. A. Yorke, "Reducing storage requirements for biological sequence comparison," *Bioinformatics*, vol. 20, no. 18, pp. 3363–3369, 2004.

[22] A. Z. Broder, "On the resemblance and containment of documents," in *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171)*, pp. 21–29, IEEE, 1997.

[23] G. Holley, R. Wittler, J. Stoye, and F. Hach, "Dynamic alignment-free and reference-free read compression," *Journal of Computational Biology*, vol. 25, no. 7, pp. 825–836, 2018.

[24] G. Marçais, D. Pellow, D. Bork, Y. Orenstein, R. Shamir, and C. Kingsford, "Improving the performance of minimizers and winnowing schemes," *Bioinformatics*, vol. 33, no. 14, pp. i110–i117, 2017.

[25] M. Belbasi, A. Blanca, R. S. Harris, D. Koslicki, and P. Medvedev, "The minimizer jaccard estimator is biased and inconsistent," *bioRxiv*, 2022.

[26] N. Dierckxsens, T. Li, J. R. Vermeesch, and Z. Xie, "A benchmark of structural variation detection by long reads through a realistic simulated model," *Genome biology*, vol. 22, no. 1, pp. 1–16, 2021.

[27] P. Biosciences, "PacBio Real-world HiFi long reads for *O. sativa*." https://downloads.pacbcloud.com/public/dataset/Sequel-IIe-202104/rice/, 2021 (last date accessed: Aug 2022).

[28] D. A. Benson, M. Cavanaugh, K. Clark, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell, and E. W. Sayers, "Genbank," *Nucleic acids research*, vol. 41, no. D1, pp. D36–D42, 2012.

[29] W. Huang, L. Li, J. R. Myers, and G. T. Marth, "Art: a next-generation sequencing read simulator," *Bioinformatics*, vol. 28, no. 4, pp. 593–594, 2012.

[30] I. Korf, M. Yandell, and J. Bedell, *Blast*. " O'Reilly Media, Inc.", 2003.