2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)

ParLECH: Parallel Long-Read Error Correction with Hadoop

Arghya Kusum Das, Kisung Lee, Seung-Jong Park
Division of Computer Science and Engineering
School of Electrical Engineering and Computer Science
Center for Computation and Technology
Louisiana State University
Baton Rouge, Louisiana 70803
Email: {adas7, klee, sjpark}@lsu.edu

***Abstract*—Long-read sequencing is emerging as a promising sequencing technology because it can tackle the short length limitation of second-generation sequencing, which has dominated the sequencing market in past years. However, it has substantially higher error rates compared to short-read sequencing (e.g., 13% vs. 0.1%), and its sequencing cost per base is typically more expensive than that of short-read sequencing. To address these limitations, we present a distributed hybrid error correction framework, called ParLECH, that is scalable and cost-efficient for PacBio long reads. For correcting the errors in the long reads, ParLECH utilizes the Illumina short reads that have the low error rate with high coverage at low cost. To efficiently analyze the high-throughput Illumina short reads, ParLECH is equipped with Hadoop and a distributed NoSQL system. To further improve the accuracy, ParLECH utilizes the k-mer coverage information of the Illumina short reads. Specifically, we develop a distributed version of the widest path algorithm, which maximizes the minimum k-mer coverage in a path of the de Bruijn graph constructed from the Illumina short reads. We replace an error region in a long read with its corresponding widest path. Our experimental results show that ParLECH can handle large-scale real-world datasets in a scalable and accurate manner. Using ParLECH, we can process a 312 GB human genome PacBio dataset, with a 452 GB Illumina dataset, on 128 nodes in less than 29 hours.**

I. Introduction

Since the start of the Human Genome Project, there have been tremendous technological advancements in sequencing technologies to understand the complexity and diversity of genomes. Particularly, the continuous evolution of Next-Generation Sequencing (NGS) technologies has drastically reduced the cost of sequencing a whole genome in recent years, and the sequence data are exploding in size and quantity. The short-read sequencing platforms, often called second-generation sequencing, have dominated the genome sequencing market in recent years by generating a huge number of short reads at a significantly reduced cost and a high throughput. Even though the short-read sequencing platforms are being widely used and potentially good for understanding the complex phenotypes, their critical limitation is that, because of their short lengths of reads, they are insufficient to resolve many long repetitive elements that are common in various genomes (e.g., eukaryotic genomes) [1].

To address this major limitation, long-read sequencing platforms, often called third-generation sequencing, aim to generate substantially longer reads (in tens or hundreds of kilobases) through direct sequencing of single DNA molecules. The long-read sequencing platforms can be useful for various research and clinical fields because they have great potential to reveal repetitive or complex regions in a single read and can support real-time sequencing using a portable device such as the pocket-sized MinION sequencer of Oxford Nanopore Technologies[1]. However, they have substantially higher error rates compared to short-read sequencing (e.g., 13% vs. 0.1%), and their sequencing cost per base is typically more expensive than that of short-read sequencing.

To address these limitations, we present a distributed hybrid error correction framework, called ParLECH, that is scalable and cost-efficient for PacBio long reads. For correcting the errors in the long reads, ParLECH utilizes the Illumina short reads that have the low error rate with high coverage at low cost. Through this hybrid error correction, we can reduce the coverage requirement for long reads and thus their sequencing cost. Specifically, ParLECH builds a de Bruijn graph (DBG) from the Illumina short reads and uses the k-mer coverage information of the short reads to correct the PacBio long reads. Unlike other DBG-based long-read error correction tools such as LoRDEC [2] and Jabba [3], we develop a distributed version of the widest path algorithm, which maximizes the minimum k-mer coverage between source and destination vertices in the DBG, to improve the accuracy of our error correction. For example, ParLECH correctly aligns 92% of the PacBio base pairs while LoRDEC aligns 86% of them for an E. Coli genome dataset.

We develop ParLECH using Hadoop MapReduce and a distributed NoSQL system, called Hazelcast. By effectively utilizing the big data frameworks, ParLECH can scale to large-scale real-world human genome data on top of hundreds of compute nodes. Our experimental results show that ParLECH can correct a 312 GB human genome PacBio dataset, with a DBG built from a 452 GB Illumina dataset (64x coverage), on 128 nodes in less than 29 hours. To the best of our knowledge,

[1]https://nanoporetech.com/products/minion

ParLECH is the first hybrid error correction framework that can handle long reads on that scale.

The rest of the paper is organized as follows. We discuss the related work in Section II and describe our error correction approach in Section III. In Section IV, we present the distributed architecture of ParLECH. Lastly, we show our experimental results in Section V and conclude this paper in Section VI.

II. Related Work

The second-generation sequencing platforms, such as Illumina sequencers, generate short reads with low error rates (e.g., 0.1%), and most of the errors are substitutions [1]. Since they can generate a huge number of short reads at a significantly reduced cost and a high throughput, many error correction tools without using any reference have been developed based on a fact that the k-mers resulting from an error base will have a significantly lower coverage compared to the actual k-mers, such as Quake [4], Reptile [5], Hammer [6], RACER [7], Coral [8], Lighter [9], Musket [10], Shrec [11], DecGPU [12], Echo [13], and ParSECH [14]. In addition to the error correction tools, there are many genome assembly tools for large-scale short reads including MPI-based tools (e.g., SWAP [15]), Hadoop-based tools (e.g., GiGA [16]), extreme-scale assemblers (e.g., HipMer [17], Lazer [18]), and GPU-accelerated tools (e.g., LaSAGNA [19]).

The third-generation sequencing platforms, such as PacBio and Oxford Nanopore sequencers, produce long reads with high error rates (e.g., 13%), and indel (insertion/deletion) errors are dominant in them. For example, indel errors are about 15 times more common than substitution errors in PacBio long reads [1]. Therefore, existing error correction tools that are designed for substitution errors in short reads are typically not adequate for long-read error correction. Another challenge for long-read error correction is the high sequencing cost per base for long reads so it would be prohibitive to get long reads with high coverage that is essential for error correction without reference genomes.

There are a few stand-alone error correction tools for long reads without any complementary short reads such as LorMA [20] and Canu [21]. Even though Canu can reduce the coverage requirement by using the tf-idf weighting scheme for long reads, the high sequencing cost for long reads can be a major obstacle to utilizing such tools for large-scale genomes. To address this limitation, several hybrid error correction tools have been developed by utilizing the complementary low-cost and high-quality Illumina short reads for correcting long reads. LoRDEC [2] builds a DBG from Illumina short reads and corrects the error regions in long reads through the local assembly on the DBG. Jabba [3] also presents a DBG-based approach by iteratively using different k-mer sizes to polish the unaligned regions of the long reads. Some hybrid error correction tools, such as PacBioToCA [22] and LSC [23], propose alignment-based approaches in which the short reads are first mapped to the long reads to create an overlap graph, and then the long reads are corrected through a consensus-based algorithm. ColorMap [24] utilizes the Dijkstra's shortest path algorithm by keeping the information of consensual dissimilarity on each edge. Proovread [25] reaches the consensus through the iterative alignment procedures by incrementally increasing the sensitivity of the long reads.

In this paper, we present ParLECH that is equipped with DBG-based error correction techniques. Unlike existing hybrid tools, we develop ParLECH as a distributed framework, so it can scale to large-scale real-world genomes. To improve the error correction accuracy, we also propose the distributed version of the widest path algorithm by utilizing the k-mer coverage information of the short reads during the local assembly.

III. Error Correction Methodology

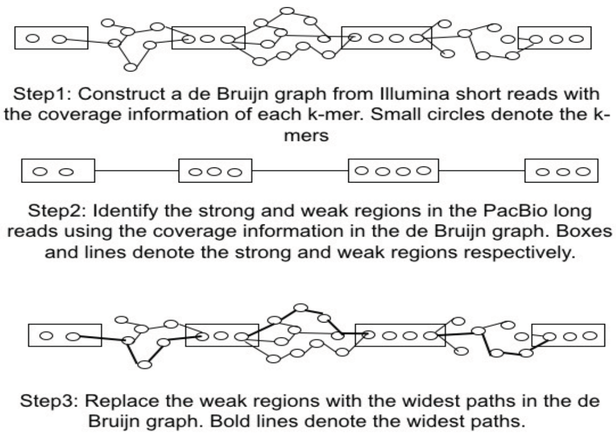

Step1: Construct a de Bruijn graph from Illumina short reads with the coverage information of each k-mer. Small circles denote the k-mers
Step2: Identify the strong and weak regions in the PacBio long reads using the coverage information in the de Bruijn graph. Boxes and lines denote the strong and weak regions respectively.
Step3: Replace the weak regions with the widest paths in the de Bruijn graph. Bold lines denote the widest paths.

Fig. 1: Error Correction Steps in ParLECH

Figure 1 shows the overview of the ParLECH's error correction approach. Inspired by other hybrid error correction tools, ParLECH also builds a DBG from the Illumina short reads to correct the PacBio long reads. Specifically, ParLECH constructs a DBG from the Illumina short reads with the coverage information of each k-mer (Step 1). Next, ParLECH partitions a PacBio long read into solid and weak regions based on the k-mer coverage information in the DBG (Step 2). Lastly, ParLECH selects boundary k-mers as either source or destination vertices in the DBG and finds the widest path for each source-destination k-mer pair, which maximizes the minimum coverage of the k-mers in the path. ParLECH replaces the weak regions in the long read with the k-mers in the widest paths (Step 3).

ParLECH chooses the widest paths for improving the accuracy based on our assumption that the probability of having the k-mer with the minimum coverage is higher in a path generated from a read with sequencing errors than a path generated from a read without sequencing errors for the same region in a genome, as discussed in [4]. For example, let R_1 and R_2 denote two short reads representing the same region in a genome, and let us assume that R_1 includes one error base and R_2 has no error base. Since ParLECH creates a DBG with the coverage information of each k-mer, the DBG representing

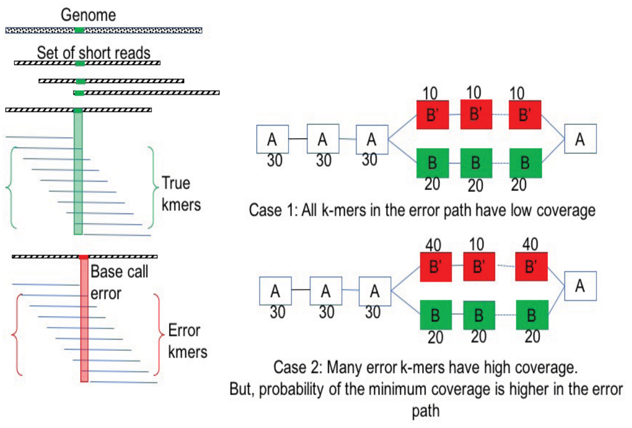

Genome
Set of short reads
True kmers
Base call error
Error kmers
10
10
10
B'
B'
B'
A
30
A
30
A
30
B
20
B
20
B
20
A
Case 1: All k-mers in the error path have low coverage
40
10
40
B'
B'
B'
A
30
A
30
A
30
B
20
B
20
B
20
A
Case 2: Many error k-mers have high coverage.
But, probability of the minimum coverage is higher in the error path

Fig. 2: Widest Path Example

R_1 and R_2 will have a fork structure leading to two different paths as illustrated in Figure 2. Based on our assumption, the probability of having the k-mer with the minimum coverage will be higher in the path generated from R_1 than the path generated from R_2. We describe each step in detail in the following section.

IV. ParLECH's Distributed Architecture

In this section, we present the distributed architecture of ParLECH.

A. Background

Error correction for sequencing data is not only data- and compute-intensive but also search-intensive because the size of the k-mer spectrum increases almost exponentially with the increasing value of k (i.e., up to 4^k unique k-mers), and we need to search in the huge search space. Since existing hybrid error correction tools are not designed for large-scale genome sequence data such as human genomes, we design ParLECH as a scalable and distributed framework equipped with Hadoop and Hazelcast. Hadoop was originally developed as the open-source counterpart of Google's MapReduce. Hadoop typically reads the input data from the underlying Hadoop Distributed File System (HDFS) in the form of data blocks. In the MapReduce abstraction, a user-defined map function is applied to each record in the input data to generate intermediate key-value pairs. In this map phase, multiple map tasks are independently running to benefit from parallel and distributed computing. The intermediate key-value pairs are sorted and grouped by unique keys and then sent to the reduce tasks. Lastly, a user-defined reduce function is applied to the grouped values for each unique key, and the final output data are typically written to HDFS.

Hazelcast [26] is an open-source distributed in-memory NoSQL database (or a key-value store). In a Hazelcast cluster, data are evenly distributed among the nodes using the MurmurHash, allowing horizontal scaling both in terms of available storage space and processing power. Hazelcast provides a hash table-like functionality, such as get and put, to insert and retrieve records in $O(1)$ time. Because of its operational similarity with a hash table, we will use a distributed NoSQL system or distributed hash table interchangeably to refer to Hazelcast. Hazelcast creates multiple in-memory instances of a hash table over multiple nodes and enables communication and load balancing among all these instances. The $O(1)$ search time of Hazelcast makes the search process in ParLECH more efficient and scalable.

B. Error Correction Pipeline

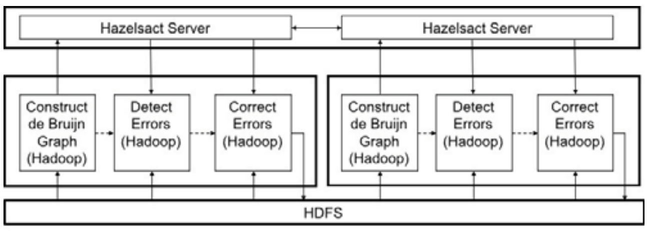
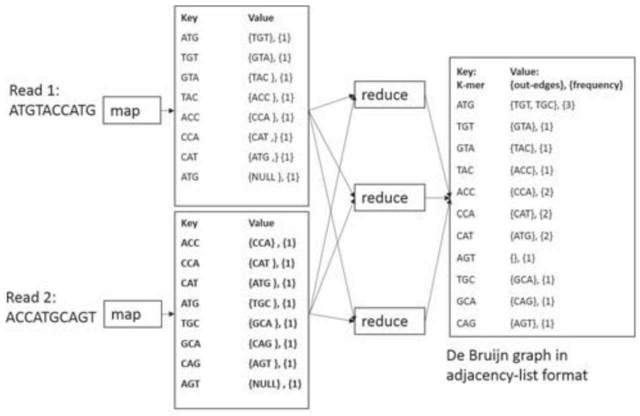


Fig. 3: Error Correction Pipeline in ParLECH

Figure 3 shows the error correction pipeline of ParLECH. It consists of three phases: 1) constructing a de Bruijn graph, 2) locating errors in long reads, and 3) correcting the errors. ParLECH uses HDFS to store the raw short and long reads as the input for the pipeline and Hazelcast to store the constructed de Bruijn graph in distributed memory. In the first phase, ParLECH uses only Hadoop to construct a de Bruijn graph from the Illumina short reads. In the subsequent phases, ParLECH uses both Hadoop and Hazelcast to locate and correct the errors in the PacBio long reads. Lastly, ParLECH writes the corrected reads into HDFS. We describe each phase in detail in the subsequent subsections.

C. Constructing a De Bruijn Graph



Key	Value
ATG	{TGT}, {1}
TGT	{GTA}, {1}
GTA	{TAC }, {1}
TAC	{ACC }, {1}
ACC	{CCA }, {1}
CCA	{CAT ,} {1}
CAT	{ATG ,} {1}
ATG	{NULL }, {1}

Key	Value
ACC	{CCA} , {1}
CCA	{CAT }, {1}
CAT	{ATG }, {1}
ATG	{TGC }, {1}
TGC	{GCA }, {1}
GCA	{CAG }, {1}
CAG	{AGT }, {1}
AGT	{NULL} , {1}

Key: K-mer	Value: {out-edges}, {frequency}
ATG	{TGT, TGC}, {3}
TGT	{GTA}, {1}
GTA	{TAC}, {1}
TAC	{ACC}, {1}
ACC	{CCA}, {2}
CCA	{CAT}, {2}
CAT	{ATG}, {2}
AGT	{}, {1}
TGC	{GCA}, {1}
GCA	{CAG}, {1}
CAG	{AGT}, {1}

Fig. 4: De Bruijn Graph Construction

ParLECH constructs a de Bruijn graph from the Illumina short reads using a Hadoop MapReduce job. In the map phase of the job, ParLECH divides each read into several short fragments (i.e., k-mers) and generates each pair of adjacent k-mers as a key-value pair in which the key (i.e., the first k-mer) denotes a vertex in the graph, and the value (i.e., the second k-mer) represents an outgoing edge from the key. In addition, the value also contains an integer value 1 that will be used to calculate the coverage of the key during the reduce phase. During the shuffle phase, all edges and count values from the same vertex (i.e., k-mer) will be collected at the same reduce task. In the reduce phase, ParLECH aggregates all edges and adds all count values for each k-mer as its coverage. Lastly, ParLECH stores the final results (i.e., the constructed de Bruijn graph) in a hash table using Hazelcast in which a key denotes a k-mer, and its value stores the coverage and outgoing edges of the k-mer. Figure 4 shows an example of this graph construction phase when k is 3.

D. Locating errors in long reads

ParLECH locates error regions in long reads based on the constructed de Bruijn graph using a Hadoop Map-only job. Specifically, ParLECH scans each long read and generates its k-mers using the same k value used in the previous phase. For each k-mer in the long read, ParLECH searches its coverage information included in the de Bruijn graph using Hazelcast. If the coverage is greater than a predefined threshold, ParLECH marks the k-mer as strong. Otherwise, ParLECH marks it as weak. If two or more adjacent k-mers are marked as weak (or strong), we denote them as a weak (or strong) region. ParLECH corrects the weak regions as described in the next subsection.

E. Correcting errors in long reads

Algorithm 1 Widest Path

```
 1: procedure MODIFIEDDIJKSTRA($Graph$, $source$, $destination$)
 2:     for (each vertex $v$ in $Graph$) do
 3:         width[$v$] := -infinity ;
 4:         previous[$v$] := undefined ;
 5:     end for
 6:     width[$source$] := infinity ;
 7:     $Q$ := the set of all nodes in Graph ;
 8:     while ($Q$ is not empty AND $destination$ is not reached) do
 9:         $u$ := vertex in $Q$ with largest width in width[] ;
10:         remove u from Q ;
11:         if (width[u] = -infinity) then
12:             break ;
13:         end if
14:         for (each neighbor $v$ of $u$) do
15:             $alt$ := max(width[$v$], min(width[$u$], width_between($u$, $v$))) ;
16:             if $alt >$ width[$v$]: then
17:                 width[$v$] := $alt$ ;
18:                 previous[$v$] := $u$ ;
19:             end if
20:         end for
21:     end while
22: end procedure
```

Like the previous phase, ParLECH uses a Hadoop Map-only job to correct weak regions in long reads based on the constructed de Bruijn graph. Given a weak region is bounded by a pair of strong k-mers, ParLECH uses the strong k-mer pair as the source and destination vertices in the de Bruijn graph for correcting the weak region, like the LoRDEC's error correction. Any path between the two vertices denotes a sequence that can be assembled from the short reads. To choose a more accurate path (i.e., sequence), ParLECH implements a widest path algorithm that maximizes the minimum k-mer coverage of a path in the de Bruijn graph. This is based on our assumption that the probability of having the k-mer with the minimum coverage is higher in a path generated from a read with sequencing errors than a path generated from a read without sequencing errors for the same region in a genome, as explained above. In other words, even if there are some error k-mers with high coverage in a path, it is highly likely that the path includes some k-mer with low coverage that will be an obstacle to being selected as the widest path, as illustrated in Figure 2. Therefore, ParLECH is equipped with the widest path technique to find a more accurate sequence to correct the weak region in the long read.

Algorithm 1 shows our widest path algorithm implemented in ParLECH, a slight modification of the Dijkstra's shortest path algorithm using a priority queue that leads to the time complexity of $O(E \log V)$. Instead of computing the shortest paths, ParLECH traverses the graph and updates the width of each path from the source vertex as the minimum width of any edge on the path (line 15).

To efficiently compute the widest paths for large-scale data, ParLECH uses a Hadoop Map-only job in which each map task handles a set of weak regions and computes the widest path between source and destination vertices for each weak region. Since the constructed de Bruijn graph is stored in a distributed hash table using Hazelcast, multiple map tasks are running in parallel to compute the widest paths. ParLECH replaces a weak region in a long read with the sequence corresponding to the widest path found in the de Bruijn graph.

F. Correcting errors on the edges of long reads

If a weak region is in the end of a PacBio read, ParLECH finds a strong boundary k-mer in the de Bruijn graph as the start vertex. However, ParLECH cannot find the strong k-mer on the other side of the weak region because the region is on the edge of the long read. To mitigate this problem, if the strong k-mer is the part of a chain structure of vertices (i.e., a path where each vertex has exactly one incoming edge and one outgoing edge), ParLECH traverses the entire chain after the strong k-mer until ParLECH detects a fork structure in the de Bruijn graph and then replaces the weak region with the traversed path. Similarly, if a weak region is in the beginning of a PacBio read, ParLECH finds a strong boundary k-mer in the de Bruijn graph as the destination vertex. Next, ParLECH traverses the chain structure, if any, before the strong k-mer and replaces the weak region with the traversed path. If the strong k-mer is not the part of a chain structure of vertices, ParLECH discards the weak region in the PacBio long reads.

V. Evaluation

In this section, we show the experimental results of ParLECH using various real-world sequence datasets.

TABLE I: Datasets

Data	Accn. #		#Reads		Data size (GB)		Read length		#Reads aligned	
	PacBio	Illumina	PacBio	Illumina	PacBio	Illumina	PacBio (Avg)	Illumina	PacBio	Illumina
E. coli	DevNet	ERR022075	282394	45440200	1.032	13.50	1120	101	78.97	99.44
Yeast	DevNet	SRR567755	2315594	4503422	0.53	1.20	5874	101	82.12	93.75
Fruit fly	BergmanLab	ERX645969	6701498	179363706	55	59	4328	101	51.14	95.56
Human	DevNet	SRX016231	23897260	1420689270	312	452	6587	101	72.3	79.60

TABLE II: Effects of Different Traversal Algorithms

Data	Methodology	#Reads	#Bases	#Aligned Reads	#Aligned bases	%Aligned reads	%Aligned bases
E. coli	ParLECH$_{WP}$	282394	309367145	264574	285070391	93.69	92.15
	ParLECH$_{SP}$	282394	307987923	247227	266373078	87.55	86.49
	ParLECH$_{Greedy}$	282394	328966341	216543	233312807	76.68	70.92
Yeast	ParLECH$_{WP}$	231594	1389446261	199332	1240945939	86.07	89.31
	ParLECH$_{SP}$	231594	1355153783	196669	1171490123	84.92	86.44
	ParLECH$_{Greedy}$	231594	1399628927	175478	1045262567	75.77	74.68
Fruit fly	ParLECH$_{WP}$	6701498	30117416348	4417627	18799138439	65.92	62.42
	ParLECH$_{SP}$	6701498	30193752318	3654326	14919815143	54.53	49.41
	ParLECH$_{Greedy}$	6701498	32131749687	2946734	12030871508	43.97	37.44

TABLE III: Accuracy Comparison (Alignments)

Data	Methodology	#Reads	#Bases	#Aligned Reads	#Aligned bases	%Aligned reads	%Aligned bases
E. coli	Original	282394	316367409	223017	237497013	78.97	75.07
	LoRDEC	282394	307987923	247227	266373078	87.55	86.49
	ParLECH	282394	309367145	264574	285070391	**93.69**	**92.15**
Yeast	Original	231594	1360457697	190184	1206524663	82.12	88.69
	LoRDEC	231594	1345253694	196669	1171490123	84.92	87.08
	ParLECH	231594	1389446261	199332	1240945939	**86.07**	**89.31**
Fruit fly	Original	6701498	29007475325	3427146	13355041639	51.14	46.04
	LoRDEC	6701498	30025673204	3654326	14919815143	54.53	49.69
	ParLECH	6701498	30117416348	4417627	18799138439	**65.92**	**62.42**

TABLE IV: Experimental Environment

Maximum #nodes	128
Processor	Intel IvyBridge Xeon
#cores per node	20
DRAM per node	64 GB
Disk per node	250 GB hard disk drive
Network	56 Gbps InfiniBand

A. Datasets

To evaluate ParLECH, we use four real-world datasets as shown in Table I. The first three datasets (E. coli, Yeast, and Fruit fly) are relatively small, and we use them to compare the accuracy of ParLECH with that of LoRDEC. We also use the Fruit fly dataset to analyze the various performance metrics in terms of scalability and execution times. We use the largest dataset (Human genome) to showcase the large-scale data handling capability of ParLECH for sequence data with several hundred GBs in size using more than 100 compute nodes. In our experiments, other existing tools could not produce the result for the dataset. We use 3 as the threshold for locating weak k-mers for all datasets.

B. Computing Environment

To evaluate ParLECH, we use *SuperMic*[2] HPC cluster, and Table IV summarizes its configuration. The maximum number of compute nodes we can use for a single job is 128. Each node has 20 cores, 64 GB main memory, and one 250 GB hard disk drive (HDD). Note that the main bottleneck for our Hadoop jobs running on top of disk-based HDFS is the I/O throughput because each node is equipped with only one HDD. We expect that the performance of ParLECH can be significantly improved by using multiple HDDs per node and/or SSD. Our previous work [16], [27], [28] demonstrates the effects of various computing environments for large-scale data processing.

[2]http://www.hpc.lsu.edu/resources/hpc/system.php?system=SuperMIC

C. Accuracy Metrics

To evaluate the accuracy of ParLECH, we use several accuracy metrics as follows. **%Aligned reads** and **%Aligned bases** indicate how well the corrected long reads and their bases are aligned to the reference genome respectively. The percentage of bases successfully aligned to the reference genome (%Aligned bases) denotes the ratio of the total number of successfully aligned bases to the total number of bases in the reference genome. To measure both the accuracy metrics for the E. Coli, Yeast, and Fruit fly datasets, we use BLASR [29] as it bridges the long indel errors better and thus reports longer alignments. For the Human genome dataset, we use BWA-mem [30] to get the alignment results quickly. We also use the **gain** metric [5] to measure the fraction of effectively corrected errors by ParLECH. The gain is defined as

$$Gain = \frac{TP - FP}{TP + FN} \tag{1}$$

TABLE V: Accuracy Comparison (Gain)

		TP	FP	FN	%Gain
E. coli	LoRDEC	31264830	330659	4230385	87.15
	ParLECH	33229635	355464	3275190	**90.05**
Yeast	LoRDEC	322660270	8989628	62594234	81.42
	ParLECH	355708411	20037769	51642375	**82.40**
Fruit fly	LoRDEC	732799376	34190591	84891209	**85.43**
	ParLECH	785735162	37126377	97826995	84.73

where TP (true-positive) is the number of error bases that are successfully corrected, FP (false-positive) is the number of true bases that are wrongly changed, and FN (false-negative) is the number of error bases that are falsely detected as correct.

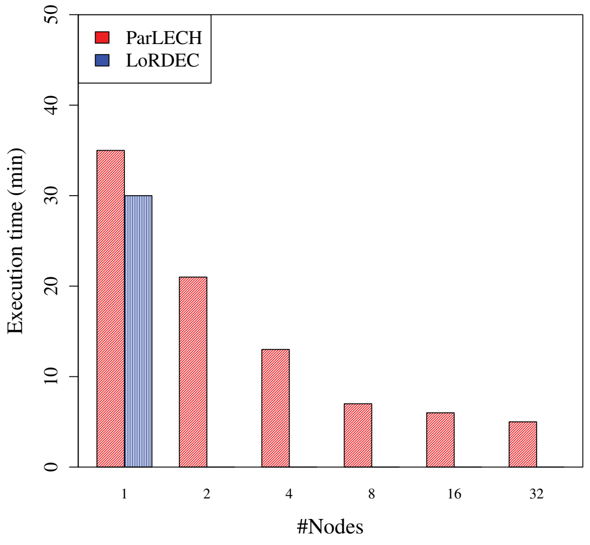
D. *Effects of Different Traversal Algorithms*

We first compare the accuracy of our widest path algorithm (ParLECH$_{WP}$ or ParLECH) with that of two baseline graph traversal algorithms, the Dijkstra's shortest path algorithm (ParLECH$_{SP}$) and a greedy traversal algorithm (ParLECH$_{Greedy}$), as shown in Table II. The Dijkstra's shortest path algorithm (ParLECH$_{SP}$) finds the path with the shortest distance between two strong boundary k-mers in the de Bruijn graph and replaces the weak region in the long read with the sequence corresponding to the shortest path. Even though the time complexity of this algorithm is similar to that of our widest path algorithm, it cannot take advantage of the k-mer coverage information included in the de Bruijn graph because it uses the same weight for all edges. On the other hand, the greedy traversal algorithm (ParLECH$_{Greedy}$) takes advantage of the k-mer coverage information included in the de Bruijn graph. As a variation of the depth first search, it traverses the de Bruijn graph from the source vertex and selects the next vertex with the maximum coverage among all neighboring vertices. However, the traversal can often end up in a tip of dead-end paths, resulting in expensive operations such as backtracking. To mitigate this problem, we use a branching factor b (100 by default) such that, after traversing b successive vertices from the source vertex, the algorithm backtracks if it cannot meet the destination vertex. The algorithm aborts when all successors from the source vertex are visited using this branching factor. We observe that ParLECH$_{WP}$ produces more accurate results than the other baseline algorithms for all datasets.

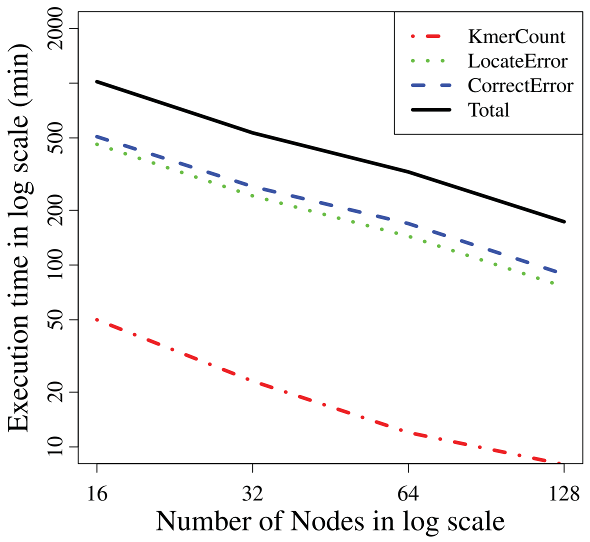
E. *Comparison with LoRDEC*

Tables III and V compare the accuracy of ParLECH with that of LoRDEC in terms of the percentage of aligned bases and gain respectively. We measure the accuracy metrics using BLASR by running multiple instances of BLASR in parallel for efficiently processing large datasets. The results demonstrate that ParLECH can generate significantly more accurate outputs for all datasets in terms of both the aligned bases and gain. Like LoRDEC, ParLECH does not try to correct the long reads in which there is no strong k-mer. However, ParLECH searches strong k-mers in all reads regardless of their length while LoRDEC filters out reads whose length is less than a threshold.

F. *Scalability*



(a) E. coli



(b) Fruit fly

Fig. 5: Scalability in ParLECH

Figure 5 demonstrates the scalability of ParLECH. On a single node for the E. Coli dataset, LoRDEC is about 16% faster than ParLECH because of the distributed computing overhead of ParLECH, as shown in Figure 5a. However, since ParLECH is developed using scalable and efficient Hadoop and Hazelcast, we can easily distribute ParLECH on a cluster with multiple compute nodes, and the performance of ParLECH improves as we add more compute nodes. On the other hand, the LoRDEC's error correction techniques including the memory- and compute-expensive de Bruijn graph construction process are not designed for distributed computing on multiple compute nodes. Even though the error correction process of LoRDEC can work on each long read independently, it does not support scheduling for such distributed computing on multiple nodes, so the scheduling should be manually implemented by users.

Figure 5b demonstrates the scalability of each phase in ParLECH for the Fruit fly dataset. The results show that the processing time of all three phases (i.e., constructing a de Bruijn graph, locating errors in long reads, and correcting errors in long reads) improves almost linearly with the increasing number of compute nodes. Therefore, the overall execution time of ParLECH also shows the almost linear scalability as we add more compute nodes.

G. Processing the large-scale human genome

TABLE VI: Correcting a Human Genome

PacBio data size	312GB
Illumina data size	452GB
#nodes used	128
Time	28.6 hours
%Aligned reads	78.3
%Aligned bases	75.43
%Gain	82.38

To showcase the large-scale data handling capability of ParLECH for sequence data with several hundred GBs in size, we report the experimental results of ParLECH for processing the Human genome dataset. This 312 GB dataset includes more than 23 million PacBio long reads with the average length of 6,587 base pairs. Its corresponding 452 GB Illumina dataset has more than 1.4 billion reads with the read length of 101 base pairs. The entire error correction process takes about 28.6 hours using 128 compute nodes, as summarized in Table VI. ParLECH correctly aligns 78.3% of reads and 75.4% of bases to the reference genome.

VI. Conclusion

In this paper, we have presented a distributed hybrid error correction framework, called ParLECH, that is scalable and cost-efficient for PacBio long reads. For correcting the errors in the long reads, ParLECH utilizes the Illumina short reads that have the low error rate with high coverage at low cost. To efficiently analyze the high-throughput Illumina short reads for correcting the long reads, ParLECH is equipped with Hadoop and Hazelcast. To improve the accuracy, based on the k-mer coverage information in the short reads, we develop the widest path algorithm, which maximizes the minimum k-mer coverage in a path of the de Bruijn graph constructed from the Illumina short reads. We replace the weak regions in a long with their corresponding widest path. Our experimental results show that ParLECH can handle large-scale real-world datasets in a scalable and accurate manner. Using ParLECH, we can process a 312 GB human genome PacBio dataset, with a 452 GB Illumina dataset, on 128 nodes in less than 29 hours. To the best of our knowledge, ParLECH is the first hybrid error correction framework that can handle long reads on that scale. As our future work, we plan to extend ParLECH for other large-scale genome analysis tasks such as hybrid genome assembly.

Acknowledgments

This work was partially funded by NSF grants (MRI-1338051, IBSS-L-1620451, SCC-1737557, RAPID-1762600), NIH grants (P20GM103458-10, P30GM110760-03, P20GM103424), LA Board of Regents grants (LEQSF(2016-19)-RD-A-08 and ITRS), and IBM faculty awards.

References

[1] S. Goodwin, J. D. McPherson, and W. R. McCombie, "Coming of age: ten years of next-generation sequencing technologies," *Nature Reviews Genetics*, vol. 17, no. 6, pp. 333–351, 2016.
[2] L. Salmela and E. Rivals, "Lordec: accurate and efficient long read error correction," *Bioinformatics*, vol. 30, no. 24, pp. 3506–3514, 2014.
[3] G. Miclotte, M. Heydari, P. Demeester, P. Audenaert, and J. Fostier, "Jabba: Hybrid error correction for long sequencing reads using maximal exact matches," in *International Workshop on Algorithms in Bioinformatics*. Springer, 2015, pp. 175–188.
[4] D. R. Kelley, M. C. Schatz, and S. L. Salzberg, "Quake: quality-aware detection and correction of sequencing errors," *Genome biology*, 2010.
[5] X. Yang, K. S. Dorman, and S. Aluru, "Reptile: representative tiling for short read error correction," *Bioinformatics*, vol. 26, no. 20, 2010.
[6] P. Medvedev, E. Scott, B. Kakaradov, and P. Pevzner, "Error correction of high-throughput sequencing datasets with non-uniform coverage," *Bioinformatics*, vol. 27, no. 13, 2011.
[7] L. Ilie and M. Molnar, "Racer: Rapid and accurate correction of errors in reads," *Bioinformatics*, 2013.
[8] L. Salmela and J. Schröder, "Correcting errors in short reads by multiple alignments," *Bioinformatics*, vol. 27, no. 11, 2011.
[9] L. Song, L. Florea, and B. Langmead, "Lighter: fast and memory-efficient sequencing error correction without counting," *Genome biology*, vol. 15, no. 11, 2014.
[10] Y. Liu, J. Schröder, and B. Schmidt, "Musket: a multistage k-mer spectrum-based error corrector for illumina sequence data," *Bioinformatics*, vol. 29, no. 3, 2013.
[11] J. Schröder, H. Schröder, S. J. Puglisi, R. Sinha, and B. Schmidt, "Shrec: a short-read error correction method," *Bioinformatics*, vol. 25, 2009.
[12] Y. Liu, B. Schmidt, and D. L. Maskell, "Decgpu: distributed error correction on massively parallel graphics processing units using cuda and mpi," *BMC bioinformatics*, vol. 12, no. 1, 2011.
[13] W.-C. Kao, A. H. Chan, and Y. S. Song, "Echo: a reference-free short-read error correction algorithm," *Genome research*, vol. 21, no. 7, 2011.
[14] A. K. Das, S. Shams, S. Goswami, R. Platania, K. Lee, and s.-J. Park, "Parsech: Parallel sequencing error correction with hadoop for large-scale genome," in *Proceedings of the 9th International BICob Conference*. ISCA, 2017.
[15] J. Meng, B. Wang, Y. Wei, S. Feng, and P. Balaji, "Swap-assembler: scalable and efficient genome assembly towards thousands of cores," *BMC bioinformatics*, vol. 15, no. Suppl 9, p. S2, 2014.
[16] A. K. Das, P. K. Koppa, S. Goswami, R. Platania, and S.-J. Park, "Large-scale parallel genome assembler over cloud computing environment," *Journal of Bioinformatics and Computational Biology*, 2017.
[17] E. Georganas, A. Buluç, J. Chapman, S. Hofmeyr, C. Aluru, R. Egan, L. Oliker, D. Rokhsar, and K. Yelick, "HipMer: an extreme-scale de novo genome assembler," in *High Performance Computing, Networking, Storage and Analysis, 2015 SC-International Conference for*. IEEE, 2015, pp. 1–11.
[18] S. Goswami, A. K. Das, R. Płatania, K. Lee, and S.-J. Park, "Lazer: Distributed memory-efficient assembly of large-scale genomes," in *Big Data (Big Data), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1171–1181.
[19] S. Goswami, K. Lee, S. Shams, and S.-J. Park, "Gpu-accelerated large-scale genome assembly," in *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2018, pp. 814–824.

[20] L. Salmela, R. Walve, E. Rivals, and E. Ukkonen, “Accurate self-correction of errors in long reads using de bruijn graphs,” *Bioinformatics*, vol. 33, no. 6, pp. 799–806, 2016.

[21] 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.

[22] S. Koren, M. C. Schatz, B. P. Walenz, J. Martin, J. T. Howard, G. Ganapathy, Z. Wang, D. A. Rasko, W. R. McCombie, E. D. Jarvis *et al.*, “Hybrid error correction and de novo assembly of single-molecule sequencing reads,” *Nature biotechnology*, vol. 30, no. 7, pp. 693–700, 2012.

[23] K. F. Au, J. G. Underwood, L. Lee, and W. H. Wong, “Improving pacbio long read accuracy by short read alignment,” *PloS one*, vol. 7, no. 10, p. e46679, 2012.

[24] E. Haghshenas, F. Hach, S. C. Sahinalp, and C. Chauve, “Colormap: Correcting long reads by mapping short reads,” *Bioinformatics*, vol. 32, no. 17, pp. i545–i551, 2016.

[25] T. Hackl, R. Hedrich, J. Schultz, and F. Förster, “proovread: large-scale high-accuracy pacbio correction through iterative short read consensus,” *Bioinformatics*, vol. 30, no. 21, pp. 3004–3011, 2014.

[26] M. Johns, *Getting Started with Hazelcast*. Packt Publishing Ltd, 2015.

[27] A. K. Das, S.-J. Park, J. Hong, and W. Chang, “Evaluating different distributed-cyber-infrastructure for data and compute intensive scientific application,” in *IEEE International Conference on Big Data*, 2015.

[28] A. K. Das, J. Hong, S. Goswami, R. Platania, K. Lee, W. Chang, S.-J. Park, and L. Liu, “Augmenting amdahl’s second law: A theoretical model to build cost-effective balanced hpc infrastructure for data-driven science,” in *Cloud Computing (CLOUD), 2017 IEEE 10th International Conference on*. IEEE, 2017, pp. 147–154.

[29] M. J. Chaisson and G. Tesler, “Mapping single molecule sequencing reads using basic local alignment with successive refinement (blasr): application and theory,” *BMC bioinformatics*, vol. 13, no. 1, p. 238, 2012.

[30] H. Li and R. Durbin, “Fast and accurate short read alignment with burrows–wheeler transform,” *Bioinformatics*, vol. 25, no. 14, pp. 1754–1760, 2009.