Efficient Distributed Smith-Waterman Algorithm Based on Apache Spark

Bo Xu, Changlong Li, Hang Zhuang, Jiali Wang, Qingfeng Wang, Xuehai Zhou
School of Computer Science and Technology
University of Science and Technology of China
Hefei, China
{xubo245, liclong, zhuangh, ustcwjl, qfwangyy}@mail.ustc.edu.cn, xhzhou@ustc.edu.cn

***Abstract*—The Smith-Waterman algorithm, which produces the optimal local alignment between pairwise sequences, is universally used as a key component in bioinformatics fields. It is more sensitive than heuristic approaches, but also more time-consuming. To speed up the algorithm, Single-Instruction Multiple-Data (SIMD) instructions have been used to parallelize the algorithm by leveraging data parallel strategy. However, SIMD-based Smith-Waterman (SW) algorithms show limited scalability. Moreover, the recent next-generation sequencing machines generate sequences at an unprecedented rate, so faster implementations of the sequence alignment algorithms are needed to keep pace. In this paper, we present CloudSW, an efficient distributed Smith-Waterman algorithm which leverages Apache Spark and SIMD instructions to accelerate the algorithm. To facilitate easy integration of distributed Smith-Waterman algorithm into third-party software, we provide application programming interfaces (APIs) service in cloud. The experimental results demonstrate that 1) CloudSW has outstanding performance and achieves up to 3.29 times speedup over DSW and 621 times speedup over SparkSW. 2) CloudSW has excellent scalability and achieves up to 529 giga cell updates per second (GCUPS) in protein database search with 50 nodes in Aliyun Cloud, which is the highest performance that has been reported as far as we know.**

Keywords-Distrubuted Smith–Waterman algorithm; SIMD instructions; Apache Spark; Scalability; Alluxio; HDFS;

I. INTRODUCTION

The Smith-Waterman (SW) algorithm [1] is a dynamic programming–based approach to identify regions of high similarity between query and subject sequences. This algorithm is the most accurate method for local sequence alignment [2]. In the past 30 years, due to its maximal sensitivity for local alignment, SW algorithm has been universally used in bioinformatics fields, including multiple sequence alignment, read mapping, biological sequence database searching, and variation detection. Nowadays, it has become a fundamental and critical component of many biological tools, such as BWA [3-5], CUSHAW2 [5], Bowtie2 [6], and DIAMOND [7].

Unfortunately, SW algorithm requires a large amount of computation due to high computational complexity. To reduce the execution time, the scientific community has made great efforts in a wide variety of hardware platforms. Among these platforms, CPU-based algorithms are most frequently used and usually use single instruction multiple data (SIMD) technology to speed up SW algorithm. Rognes and Seeberg [8] were the first to take advantage of multimedia extensions (MMX) and streaming SIMD extensions (SSE) technology. Farrar [9] presented a new SW implementation where the SIMD registers are parallel to the query sequence with a striped access pattern. Zhao [10] presented SSW, a library developed in C/C++ to facilitate SW integration to other genomic applications. Daily [11] presented Parasail, a C-based library containing different sequence alignment algorithms. These implementations take advantage of parallelization strategies. However, they show limited scalability.

With the development of next-generation sequencing (NGS) technology, the cost of sequencing drop faster than Moore's law, and the number of newly sequenced data has exhibited an exponential increase in recent years [12]. In the current scenario where data growth is in petabytes a day [13]. It has become a huge challenge that how to store and analyze the big data of sequences. So faster implementations of the sequence alignment algorithms are needed to keep pace. Zhao [14] implemented the SW algorithm on Apache Spark for the first time, which was called as SparkSW. To the best of our knowledge, DSW [15] was the fastest publicly available solution of the exact SW algorithm on Apache Spark before this paper and achieved up to 201 times speedup over SparkSW.

In this paper, we present a more efficient distributed Smith-Waterman algorithm, CloudSW, which leverages Spark and SIMD instructions to accelerate SW algorithm. The major contributions of this paper are as follows:

- We leverage Spark to enable conventional SIMD-based algorithms run in a horizontally scalable distributed environment and it returns detailed alignment information with very small performance loss, including optional score and alignment traceback.
- Using query profile move forward strategy to reduce repeated calculation time of query profile and avoiding the costly memory copy by swapping pointer references to an auxiliary column.
- For the top k scenario, we first calculate the alignment score of all sequences and return top k score, then calculate the traceback of alignment, which reduce the traceback calculate time of most sequence alignment.

The rest of this paper is organized as follows. Section Ⅱ provides the background and related work on distributed SW algorithm. In Section Ⅲ, we describe design of CloudSW. Section Ⅳ shows our experiments and evaluation results. In Section Ⅴ, we summarize the conclusions and discuss future research work.

2159-6190/17 $31.00 © 2017 IEEE
DOI 10.1109/CLOUD.2017.83

II. BACKGROUD AND RELATED WORK

A. Overview of the Smith-Waterman Algorithm

The SW algorithm is exploited to identify the optimal local alignment between query and subject sequences by computing the similarity score with methods of dynamic programming.

There are two types of gap penalty model, known as linear-gap model and affine-gap model [16]. The linear-gap model is a simple model with constant gap penalty. The affine-gap model has open and extension gap penalties. The penalty for opening a gap is normally bigger than the penalty for extending a gap, which is more biologically realistic as few continuous gaps are expected to occur frequently than discrete single gap in practice.

The SW algorithm can be divided into two stages: (1) similarity matrix (also called alignment matrix) filling, to obtain optimal alignment score; and (2) traceback, to obtain optimal alignment path.

1) Similarity Matrix Filling

The query sequence is defined as $Q = q_1 q_2 q_3 \dots q_m$ and its length is $length(Q) = \mathrm{m}$. The subject sequence is defined as $S = s_1 s_2 s_3 \dots s_n$ and its length is $length(S) = \mathrm{n}$. SM is the substitution matrix which defines the substitution scores for all residue pairs. Usually the weight $SM(q_i, s_j) \leq 0$ when $q_i \neq s_j$ and $SM(q_i, s_j) > 0$ when $q_i = s_j$. Common substitution matrices for protein alignment are BLOSUM or PAM families. The penalty for opening a gap and extending a gap are defined as G_{open} and G_{ext}, respectively. $E_{i,j}$ and $F_{i,j}$ are the scores involving the first i symbols of Q and the first j symbols of S, and ending with a gap in sequence Q or S, respectively [8]. $H_{i,j}$ is the alignment score, which are shown as follows.

$$E_{i,j} = \max\{E_{i,j-1} - G_{ext}, H_{i,j-1} - G_{open}\} \quad (1).$$

$$F_{i,j} = \max\{F_{i-1,j} - G_{ext}, H_{i-1,j} - G_{open}\} \quad (2).$$

$$H_{i,j} = \max\{0, E_{i,j}, F_{i,j}, H_{i-1,j-1} - SM(q_i, s_j)\} \quad (3).$$

The values of $E_{i,j}$, $F_{i,j}$, and $H_{i,j}$ are initialized with 0 when $i = 0$ or $j = 0$. The maximal alignment score in the matrix H is the optimal local alignment score.

2) Traceback

Traceback is quite important for subsequent analysis or optimization, such as read mapping and variation analysis. After optimal local alignment score is obtained, a traceback procedure is executed to obtain the pair of segments with maximum similarity based on matrix H, until a position whose value is zero is reached. These two segments represent the best local alignment and are usually described as CIGAR strings [17].

SW algorithm can be divided into two modes according to whether there is a traceback: the faster mode only computes the alignment scores (ASM) and another mode supports alignment traceback (ATM) [18].

Fig. 1 illustrates the calculation process of Smith–Waterman algorithm between query sequence PNHIGD and subject sequence PTHIKWGD, which uses BLOSUM50 as substitution matrix, 12 as open gap penalty and 2 as extension gap penalty. The subject sequence is P18691 from the public datasets UniRef100 [19]. After similarity matrix filling, we obtain optimal local alignment score is 27. Then executing traceback procedure and the traceback of query sequence is described as 4M2D2M and starts with 0 in subject sequence.

	-	P	T	H	I	K	W	G	D
-	0	0	0	0	0	0	0	0	0
P	0	10	0	0	0	0	0	0	0
N	0	0	10	1	0	0	0	0	2
H	0	0	0	20	8	6	4	2	0
I	0	0	0	8	25	13	11	9	7
G	0	0	0	6	13	23	11	19	8
D	0	0	0	4	11	12	18	10	27

Best Local Alignment PNHI - - GD
PTHIKWGD

Figure 1. Illustration of Smith–Waterman algorithm between sequences PNHIGD and PTHIKWGD.

B. Smith-Waterman SIMD Implementation

SIMD technology has been introduced to reduce SW execution time by leveraging data parallel strategy at the instruction level, which has been demonstrated to be efficient. There are some important technologies in this field as follows.

1) Query Profile Technique and Striped Access Pattern

When calculating $H_{i,j}$, the value from the substitution matrix $SM(q_i, s_j)$ is added to $H_{i-1,j-1}$. To avoid the lookup of $SM(q_i, s_j)$ for each cell, Rognes and Seeberg [8] calculated a query profile parallel to the query for each possible residue, which consists in building an auxiliary two-dimensional array of size $|\mathrm{Q}| \times |\Sigma|$, where Q is the query sequence and Σ is the alphabet. Each row of this array contains the scores of the corresponding query residue against each possible residue in the alphabet. Farrar [9] vectorized along the query sequence and the alignment matrix computations were reorganized to do them in a striped manner. The query is divided into equal length segments. The number of segments, p, is equal to the number of elements being processed in the SIMD register. The length of each segment, t, is $(\mathrm{length(Q)} + \mathrm{p} - 1)/\mathrm{p}$. For 128-bit wide registers, When processing byte integers (8 bit values) the p=16 and when processing word integers (16 bit values) the p=8. The query profile (QP) are defined as (4), where $1 \leq \mathrm{i} \leq \mathrm{length(Q)}$.

$$QP_{i,j} = SM(q_{((i-1) \bmod p)*t+(\lfloor \frac{i}{p} \rfloor + 1)}, s_j) \quad (4).$$

Where the remainder is obtained by $(i-1)\ mod\ p$ and the quotient is obtained by $\left\lfloor \frac{i}{p} \right\rfloor$. This striped access pattern minimizes data dependences impact and reduces misaligned vector accesses [2].

2) Shifting Technique

Rognes and Seeberg [8] used 8-bit integer data, and score representation range is 0–255. Zhao [10] allowed byte(8-bit integer) or word(16-bit integer) to represent data, which can represent max score is 32767. Overflow will occur when sequences are long and/or very similar between them. Similar situation arises in DSW due to it is based on SSW. Moreover, a feature of the striped method is the recalculation of a column within the dynamic programming table until the column converges. To improve those problems, Parasail [11] added support for 32-bit and 64-bit data, and used a technique of shifting the origin of the calculation towards the smallest representable integer and used saturation arithmetic to keep the calculation from underflowing. For example, for 8-bit integers -127 is treated as 0, allowing the entire range of 0 through 255 to be utilized, which is not unlike previous SSE2 implementations that mitigated the lack of a signed 8-bit vector maximum instruction and instead used unsigned integers. What's more, Parasail has supported different instructions, including SSE, AVX, and KNC [11].

C. Apache Spark

Apache Spark [20] runs programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk. Spark has emerged as the de facto framework for big data analytics with its advanced in-memory programming model [21]. In parallel programming, Spark supplies two main abstractions: Resilient Distributed Datasets (RDDs) [22] and parallel operations on datasets. We can easily use many operations (i.e., data transformations and actions) to analysis sequence data with Spark RDDs in a horizontally scalable distributed environment.

With the decline of memory prices, in addition to memory-based computing framework has been widely used, memory-based storage technology has also been developed. Alluxio is memory-based distributed file system enabling reliable data sharing at memory speed across cluster computing frameworks [23, 24].

III. Design Overview

This section overviews the design of CloudSW. We focus on how to reduce the execution time of SW algorithm. Conventional SIMD-based algorithms are most frequently used because they are compatible with most modern x86 CPUs, and also are efficient in the single-node scenario. However, they show limited scalability. So our basic idea is using Spark to enable single-node SIMD-based algorithms to run in a horizontally scalable distributed environment and then constantly improving the performance of the algorithm.

In process of sequence alignment, it is frequent to get the most K similar pairs of the alignments, such as protein database searching and seed extension of read mapping. Hence, CloudSW is composed of four stages: Data preprocessing, the core calculations of SW by map procedures, reduce procedures and obtain traceback of alignment. Details of each phase are described below

A. Data preprocessing

The input data of CloudSW include different datasets: query sequences, subject sequences and score/substitution matrix. Conventional data formats, such as FASTA, FASTQ, and SAM, were designed for single node processing. They need to be preprocessing for distributed environment. CloudSW provides a converter for different formats. As shown in Algorithm 1, pseudocode from line 1-3 are designed for preprocessing query file, database file, and score/substitution matrix. CloudSW uses template pattern and provides APIs for users to use their own function to preprocess the data by extending the template class and override the implementation.

CloudSW support input/output data from/to two different data sources, which respectively are HDFS and Alluxio. HDFS is more popular and Alluxio can provide more read/write speed by caching hot files in memory.

When data has been preprocessed, CloudSW will cache data in memory by invoking a persisting method if the number of querys are bigger than the set threshold, and reuse these cached data recursively in the next operations.

Algorithm 1 CloudSW (ATM) on Apache Spark

```
Function 1: run(sc, queryFile, dbFile, scoreMatrix, open, ext, splitNum, topK)
Output: Array[AlignmentRecordTopK]
1:    querys ← preprocessQuery(queryFile, sc)
2:    dbRDD ← preprocessDB(dbFile, splitNum, sc)
3:    SM ← preprocessSM(scoreMatrix, sc)
4:    if querys.length > querysThreshold then
5:       dbRDD.persist(StorageLevel.MEMORY_ONLY)
6:    end if
7:    result←align(sc, querys, dbRDD, SM, open, ext, topK)
8:    return result
Function 2: align(sc, querys, dbRDD, scoreMatrix, open, ext, topK)
Output: Array[AlignmentRecordTopK]
9:     mapRDD ← dbRDD.mapPartitions{ subs =>
10:      profiles ← mediator.createProfiles(querys, scoreMatrix)
11:      subs.map{ sub =>
12:        queues ← new Array[PriorityQueue[AlignmentRecord]](querys.length)
13:        for i ← 0 to querys.length-1 do
14:          queues(i) ← new PriorityQueue[AlignmentRecord](topK)(Ordering)
15:          alignment ← mediator.ParasailAlign(profiles(i), sub, open, ext)
16:          queues(i) ++= Iterator.single(alignment)
17:        end for
18:        queues
19:        }
20:    }
21:    reduceRDD ← mapRDD.reduce { (queue1, queue2) =>
22:      for i ← 0 to queue1.length-1 do
23:        queue1 (i) ++= queue2 (i)
24:      end for
25:      queue1
26:    }
27:    asms ← reduceRDD.toArray
28:    atms ← new Array[AlignmentRecordTopK](querys.length)
29:    for i ← 0 to asms.length-1 do
30:      asm, atm ← r1(i).toArray.sorted(Ordering.reverse)
31:      for j ← 0 to asm.length-1 do
32:        if asm(j).score < SSWThreshold then
33:          atm(j) ← mediator.SSWAlign(querys(i), asm(j) , scoreMatrix, open, ext))
34:        else
35:          atm(j) ← mediator.LSW(querys(i), asm(j), scoreMatrix, open, ext)
36:        end if
37:      end for
38:      atms(i) ← new alignmentRecordTopK(querys(i), topK, atm)
39:    end for
40:    return atms
```

Annotation: (1) *sc*: SparkContext; (2) *queryFile*: query file, (3) *dbFile*: database file; (4) *scoreMatrix*: score matrix; (5) *open*: open gap penalty; (6) *ext*: extension gap penalty; (7) *splitNum*: number of partitions; (8) *topK*: top k;

B. Map with SIMD instructions

After the data is ready, CloudSW uses SIMD technology to speed up SW algorithm in each map task. To reduce the execution time of SW, CloudSW only calculate max score between query and subject sequences as shown in Fig.2. We design query profile move forward strategy for CloudSW. Details as shown in function 2 of Algorithm 1, we first compute the query profile based on (4) by inputting query sequences and score matrix. Then the query align with every subject sequence in each reference database partition by using the same query profile, which effectively avoids repeated calculations and saves a lot of time. ASM of CloudSW avoids the costly memory copy like DSW by swapping pointer references to an auxiliary column. It is based on a SIMD-based SW algorithm Parasail, which is efficient algorithm due to leveraging shift technique and achieve the highest reported for implementations based on Farrar's striped approach in the single-node scenario [11].

However, Apache Spark cannot directly invoke SIMD-based sequence alignment algorithms which are written in C language. In order to integrate SIMD-based sequence alignment algorithms into Spark, we design a mediator based on Java Native Interface (JNI). Spark applications can indirect invoke SIMD-based sequence alignment algorithms by the mediator.

As shown in Figure 2, CloudSW with ASM will return a new RDD in map phase. The alignment results in each partition of the RDD include max score, the name of reference sequence and other properties, without traceback.

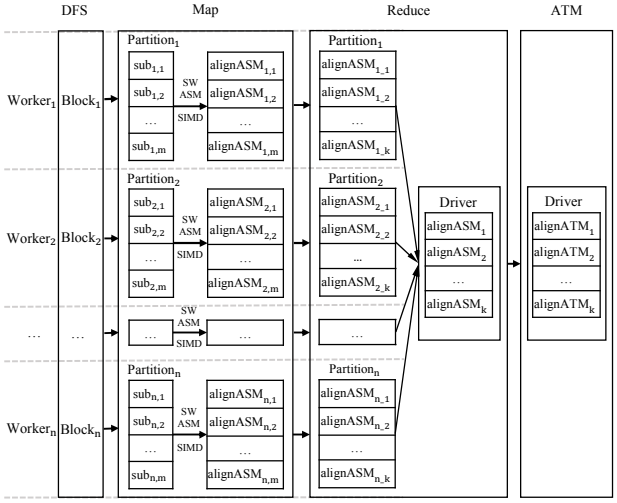

Figure 2. Overview of the CloudSW algorithm

C. Reduce with Priority Queue Array

DSW obtains the most K similar pairs of the alignments by invoking top method of Spark, which is not efficient when there are multiple query sequences in query dataset and their length is short. In order to improve performance in this scenario, CloudSW will execute reduce task after map phase as shown in Fig.2. CloudSW uses priority queue array to save top k alignment results, which is shown in function 2 of Algorithm 1. The order is based on maxScore in alignment objects and implemented by an implicit method. The reduce function aggregates global top k disordered alignment objects for each query. Finally, top k disordered alignment objects will be sorted and used for obtaining traceback in next phase.

D. Calculating the Traceback of Alignment

The results in Spark reduce phase are without traceback of alignment, which is quite important for subsequent analysis or optimization. As shown in function 2 of Algorithm 1, we align the top k ordered alignment objects with SW algorithm again. The SSW-based algorithm will overflow and return an error score when it alignment two highly similar long sequences. To overcome it, we design a threshold for it. When the score is less than the threshold, CloudSW uses a SIMD-based algorithm to obtain traceback of alignment, which is based on SSW algorithm. Otherwise CloudSW will calculate traceback of alignment by using the local SW algorithm (LSW) without SIMD instructions, which is shown in Algorithm 2. The function 1 is designed for filling similarity matrix, and builds a move matrix of sequence alignment simultaneously. Then will calculate the alignment path by tracing back with H, the coordinates of max score and the move matrix. The details are shown in Algorithm 2. The traceback path will be returned in terms of CIGAR and corresponding coordinates.

CloudSW provides APIS of ASM and ATM for third-party software in cloud.

Algorithm 2 LSW (ATM)

```
Function 1: buildMatrix(query, sub, scoreMatrix, open, ext)
Output: (maxScore, xLocation, yLocation, moveMatrix)
1:  set E, F, H, moveMatrix←Matrix(query.length+1)(sub.length+1) and initialization
2:  set maxScore, xMax, yMax←0
3:  for i ← 1 to query.length do
4:    for j ← 1 to sub.length do
5:      E(i)(j) ←max{ E(i)(j - 1) - ext, H(i)(j - 1) - open }
6:      F(i)(j) ←max{ F(i - 1)(j) - ext , H(i - 1)(j) -open }
7:      h ← H(i - 1)(j - 1) + scoreMatrix (query(i), sub (j))
8:      switch
9:        case E(i)(j) is max then H(i)(j) ←E(i)(j), moveMatrix(i)(j)← 'I' break
10:       case F(i)(j) is max then H(i)(j) ←F(i)(j), moveMatrix(i)(j)← 'J' break
11:       case h is max then H(i)(j) ←h, moveMatrix(i)(j)← 'B' break
12:       case 0 is max then H(i)(j) ←0, moveMatrix(i)(j)← 'T' break
13:     end switch
14:     if H(i)(j) > maxScore then
15:       set maxScore ←H(i)(j), xMax←I, yMax←j
16:     end if
17:   end for
18: end for
19: return (maxScore, xMax, yMax, moveMatrix)
Function 2: traceback (matrix, xMax, yMax, xC, yC)
Output: (xCigar, yCigar, xStart, yStart)
20: switch
21:   case matrix(i)(j) = 'B' then return traceback(matrix, i - 1, j - 1, xC+"M", yC+"M")
22:   case matrix(i)(j) = 'J' then return traceback(matrix, i - 1, j, xC + "I", yC + "D")
23:   case matrix(i)(j) = 'I' then return traceback(matrix, i, j - 1, xC + "D", yC + "I")
24:   case matrix(i)(j) = 'T' then return (buildCigar(xC), buildCigar(yC), i, j)
25: end switch
```

Annotation: I: move in I coordinate; J: move in J coordinate; B: move in both I and J; T: terminate move; M: match or dismatch; D: delete; I: Insert.

IV. EXPERIMENTS

In this section, CloudSW is evaluated from different aspects. First, complete description of the experimental setup is provided. Next, CloudSW is analyzed in different data sources, different modes of operation and score matrices. Then the scalability of CloudSW is evaluated by a various number of computing nodes in Aliyun E-MapReduce cluster, which consists of 50 ecs.sn2.large

nodes. Last but not least, we present a detailed performance comparison between CloudSW and state-of-the-art algorithms, including the algorithms in the single-node scenario, the Spark-based algorithms in cluster and the algorithms in different hardware platform.

A. Experimental Setup

Our experiments were performed in a local cluster in our laboratory and/or a remote cluster in the Alibaba Aliyun Cloud [25]. In the local cluster, there are 8 physical node and the operation system of each node is Ubuntu-14.04.1. Each node has a dual core Intel Xeon W3505 CPU with 22GB of RAM. The Apache Spark version is 1.5.2 and each node has 8GB memory for all Spark applications. The Alluxio version is 1.3.0 and each node has 12GB memory for Alluxio worker. In the remote cluster, we purchased 50 Aliyun Cloud virtual machines and deployed the nodes with EMR-2.3.0, which includes Hadoop 2.7.2, Yarn 2.7.2 and Spark 1.6.2. The operation system of each node is CentOS release 6.5. Each node has a four core Intel Xeon E5-2680 CPU and 16G memory. The default version of CloudSW is 2.0.6. By default, all experiments run in the local cluster.

The data used in the experiments are all from the public datasets UniProt Reference Clusters (UniRef100) [19]. The details of the experimental datasets are listed in Table I.

TABLE I. THE LIST OF SUBJECT AND QUERY SEQUENCES

D	TD	NS	Q	NQ	LQ
D1	32	77085	Q1	P18691	8
D2	64	153291	Q2	P83140	16
D3	128	317193	Q3	P20738	32
D4	256	737161	Q4	O55746	64
D5	512	1620389	Q5	Q6GZW8	128
D6	1024	2759958	Q6	Q6GZX4	256
D7	2048	4994623	Q7	Q19LI2	512
D8	4096	9639388	Q8	Q7TQI7	1024
D9	8192	20873256	Q9	Q8IYD8	2048
D10	16384	46282321	Q10	R0INU3	4096

D: databases of subject sequences; TD: the total number of symbols in database (million chars); NS: the number of subject sequences;
Q: datasets of query sequence; NQ: name of query sequence; LQ: length of query (chars); Only one query sequence in each dataset.

Where TD $= \sum_{i=1}^{NS} length(S_i)$, S_i is i-th subject sequence in database.

Cell updates per second (CUPS) is a commonly used performance measure in the Smith–Waterman context, because it allows removal of the dependence on the query sequences and the databases utilized for the different tests as well as the hardware device [26]. A CUPS represents the time for a complete computation of one cell in matrix H, including all memory operations and the corresponding computation of the values in the E and F arrays. Given a query sequence dataset Q and a database D, the GCUPS (giga cell updates per second) value is calculated by (5).

$$\text{GCUPS} = \frac{\sum_{i=1}^{M} length(Q_i) \times \sum_{j=1}^{N} length(S_j)}{t \times 10^9} \quad (5).$$

Where $\text{length}(Q_i)$ is the number of symbols in the i-th query sequence of Q and M is the total number of query sequences of Q, $\text{length}(S_j)$ is the number of symbols in the j-th subject sequence of D and N is the total number of subject sequences of D, and t is the runtime in seconds.

B. Performance Evaluation

CloudSW allows users to input data from different data sources, provides different modes of operation for them and allows users to use different score/substitution matrices, open and extension gap penalty as input parameters. Related performance evaluation are shown as follows.

1) Experiment on Different Data Sources

CloudSW support HDFS and Alluxio at present. The first experiment uses a reference database with fixed size and different length of query sequences. We select D8 as reference database and the total number of symbols in D8 is about 4096 million chars (D8). The query sequences are Q1 to Q10 (see Table I). The experiment runs CloudSW that only computes the alignment scores (ASM) with 8 nodes. This experiment uses BLOSUM50 as substitution matrix, 12 as open gap penalty and 2 as extension gap penalty.

Fig. 3(a) shows the runtime of CloudSW (ASM) input data from HDFS and Alluxio. Alluxio mode achieves up to 2.39 times speedup over HDFS mode. With the increase of query sequence length, the gap of runtime between Alluxio and HDFS modes has a tendency of decreasing gradually. This is easy to understand because the size of data is basically unchanged and the computation time increases.

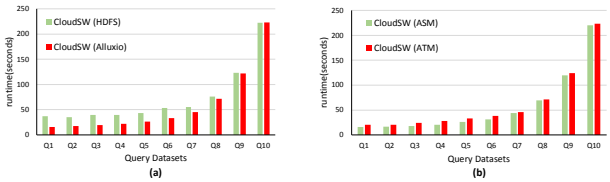

Figure 3. Performance Evaluation on Different Data Sources and Modes of Operation

2) Experiment on Different Modes of Operation

CloudSW provides two different modes of operation, ASM and ATM. ASM only computes the alignment scores, but ATM supports both alignment scores and traceback. The second experiment makes a performance evaluation on ASM and ATM in CloudSW, which both input data from Alluxio and top k is 5. The environment and other configuration of this experiment are the same as in the previous experiment.

Fig. 3(b) shows the runtime of ASM and ATM in CloudSW. The ATM is slower than ASM, which average is 0.98%. Although ATM need to compute traceback of sequence alignment and ASM don't need. The extra overhead is very small because the number of sequences that need to compute traceback are few in the scenario.

3) Experiment on Different Score Matrices, Open and Extension Gap Penalty

In this part, we use different score/substitution matrices, gap penalties as input parameters and analysis their effects for CloudSW. The environment and other configuration of this experiment are the same as in the previous experiment.

Fig. 4 shows the performance of CloudSW with different configurations. CloudSW has a better performance than others in most cases when using BLOSUM62 as substitution matrix, 12 as open gap penalty and 2 as extension gap penalty. CloudSW has a worse performance when it uses BLOSUM50 as substitution matrix, 11 as open gap penalty and 1 as extension gap penalty. Different combinations of substitution matrices and gap penalties have different effects on the computational process of CloudSW.

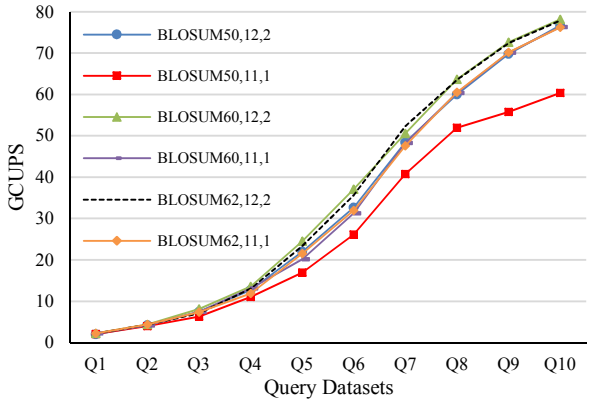

Figure 4. Performance of CloudSW in Different Substitution Matrices, Gap Penalties

C. Scalability Evaluation

In order to better evaluate the scalability of CloudSW, we use a remote cluster in the Alibaba Aliyun Cloud, which is one of the biggest commercial cloud platforms in China and provides worldwide cloud service. The information of the cluster has been introduced in experimental setup part. We use D9 as a reference database and 40 different query sequences as query dataset D_{avg}, which are selected from UniRef100 and all are 392 chars long according to the average length of the D9 (see Table I). With the development of NGS technology, the length of sequences is getting longer. To evaluate the scalability of CloudSW in long sequences, we select 4 different query sequences as query dataset D_{long}, which are selected from UniRef100 and their average length is 10240 chars. We run ATM of CloudSW with Spark on Yarn and input data from HDFS, which is more commonly used in industry.

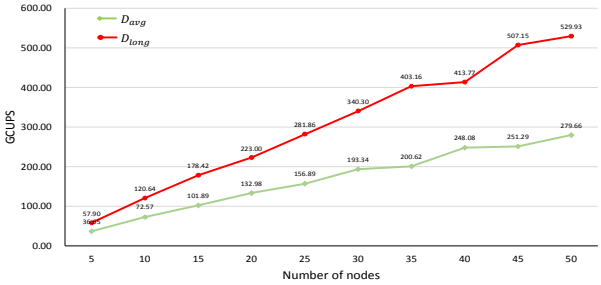

Figure 5. The GCUPS of CloudSW on the Different Number of Nodes

Fig. 5 shows the GCUPS of CloudSW on the different number of nodes. CloudSW has near linear performance improvement for D_{avg} *and* D_{long} as the number of nodes increases. The performance using dataset D_{long} is higher than that using dataset D_{avg}. The experiment demonstrates CloudSW has excellent scalability for two different datasets. CloudSW achieves up to 279.66 GCUPS with D_{avg} and 529.93 GCUPS with D_{long}, which both run in the Aliyun cluster with 50 nodes. CloudSW has the potential to achieve better performance with more nodes.

D. Performance Comparison

1) Performance Comparison with the Algorithms in the Single-node Scenario

The conventional Smith-Waterman algorithms based on SIMD are designed for running in the single-node scenario. CloudSW also can run in the single-node scenario, but its major advantages is using multi-node to reduce the execution time of SW and this is also the purpose of CloudSW. So we make a performance comparison in two scenario respectively.

Fig. 6 shows performance comparison with the single-node algorithms. All algorithms in Fig. 6(a) run with one core. CloudSW (ATM) has 3.36 times speedup over SSW and similar performance with Parasail, but Parasail do not support traceback of alignment. Fig. 6(b) shows the speedup of CloudSW with different cores and uses single-core Parasail as baseline. Each node just is used a core by CloudSW. CloudSW achieves near linear speedup over Parasail by increasing the number of cores and up to 6.60 times speedup over Parasail with 8 cores.

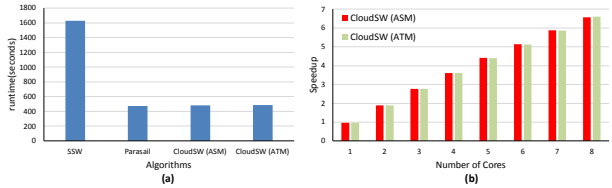

Figure 6. Performance Comparison with the Single-node Algorithms

2) Performance Comparison with the Spark-based Algorithms in Cluster

The fast growing reference database volumes and longer length of query sequence are new challenges for sequence alignment. In this part, we design two different experiments to validate the ability of CloudSW.

a) Different Length of Query Sequences

The experiment uses a reference database with fixed size and different length of query sequences. The total number of symbols in the reference database is about 4096 million chars (D8) and the query sequences are Q1 to Q10 (see Table I).

Figure 7 shows speedup of DSW and CloudSW over SparkSW for different query datasets. The experimental result shows that although DSW has significant performance improvements than SparkSW, CloudSW is faster than DSW and achieves up to 3.29 times over DSW. ASM and ATM of CloudSW both have outstanding

performance improvements and achieve up to 621.69 and 618.93 times over SparkSW respectively. SparkSW only provides ASM and no traceback. The speedup of DSW and CloudSW over SparkSW both increases with the increase of query length.

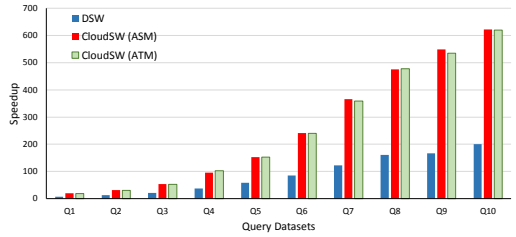

Figure 7. Speedup of DSW and CloudSW over SparkSW for Different Query Datasets

b) Different Size of Reference Databases

The experiment uses a query sequence with fixed length and different size of reference databases. The fixed length of query sequence is 512 chars (Q7), and the reference databases are D1 to D10 (see Table I). The experiment both inputs data from Alluxio.

Figure 8 shows speedup of DSW and CloudSW over SparkSW for different reference databases. The experimental result shows that CloudSW achieves average 2.13 times over DSW and 198.53 times over SparkSW. When reference databases are D1 to D9, the speedup of DSW and CloudSW over SparkSW both increases with the increase of database size. The speedup in D10 is less than that in D9 and its network overhead increases.

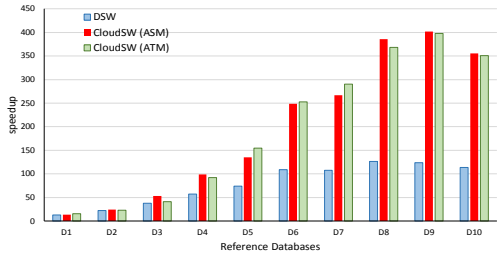

Figure 8. Speedup of DSW and CloudSW over SparkSW for Different Reference Databases

3) Performance Comparison with the Algorithms in Different Hardware Platform

In the past decades, the scientific community and industry has made great efforts to reduce the execution time in a wide variety of hardware platforms, including central processing units (CPU), graphics processing units (GPU), field programmable gate arrays (FPGA), and Xeon Phi coprocessors. Especially in recent years, SW algorithm has been greatly optimized and performance has been improved thousands of times relative to 1997 year [2]. However, the number of newly sequenced data has exhibited an exponential, which requires sequence alignment algorithms to keep pace. In bioinformatics, GCUPS is a commonly used performance measure on different hardware devices. As shown in Table Ⅱ, we simply summarize different implementations of SW algorithm on different hardware platforms, including CPU, GPU, FPGA, Xeon Phi, and hybrid platforms, which all are one of state-of-the-Art implementations/algorithms that has been reported in protein database search. Those implementations run in different hardware device and use different configurations, which are introduced in original paper and/or summarized in the relevant reviews [2, 27]. In this part, we focus on the highest GCUPS of different implementations on different hardware and don't care how much computing resources and what their type are they used, including the number of nodes, the type of CPU and memory, different parameters, etc. To the best of our knowledge, CloudSW achieves the highest performance among different implementations of SW algorithm on different hardware platforms. The highest performance is 529.9 GCUPS, which runs with ATM of CloudSW in 50 nodes cluster and takes both data input and compute time into account.

TABLE II. IMPLEMENTATION SUMARY OF SW ALGORITHM BASED ON DIFFERENT HARDWARE PLATFORMS

Year	Implementation	Hardware Platform	GCUPS
2015	SWIMM [28]	CPU	360.0
2016	Parasail [11]	CPU	291.5
2010	Khalafallah and Mahmoud [29]	GPU	66.0
2016	OSWALD [30]	FPGA	178.9
2014	SWAPHI [31]	Xeon Phi	228.4
2015	LSBDS [32]	Xeon Phi	220.0
2013	SUDASW++3.0 [33]	CPU + GPU	185.6
2010	Meng and Chaudhary [34]	CPU + FPGA	11.3
2014	Rucci and Giusti [35]	CPU +Xeon Phi	62.6
2016	HHeterSW [36]	CPU+GPU+Xeon Phi	139.7
2015	SparkSW [14]	CPU-based cluster	0.3
2017	CloudSW	CPU-based cluster	529.9

V. CONCLUSION AND FUTURE WORK

CloudSW is an efficient Spark-based distributed Smith-Waterman algorithm optimized for producing the optimal local alignment between pairwise sequences and obtaining the most K similar pairs of the alignments in a horizontally scalable distributed environment. CloudSW allows users to input data from different data sources, provides different modes of operation, ASM and ATM, and also allows users to use different configurations. Those function are provided in terms of API service in cloud for third-party software, which provides possible help for conventional biological tools running in cloud, such as BWA, Bowtie2.

The experimental results demonstrate that 1) CloudSW has outstanding performance and achieves up to 3.29 times over DSW and 621 times speedup over SparkSW. 2) CloudSW has excellent scalability and achieves up to 529 giga cell updates per second (GCUPS) in protein database search with 50 nodes in Aliyun Cloud. CloudSW is also faster than conventional efficient SIMD-based algorithm, SSW, and has similar performance with Parasail, but Parasail do not support traceback of alignment.

In the future, we plan to explore different technologies to improve performance in more node cluster, and optimize for different type of sequence data.

ACKNOWLEDGMENT

We would like to thank technicians of Alibaba Aliyun Cloud for help. This work was supported by the National

Science Foundation of China (No. 61379040), Anhui Provincial Natural Science Foundation (No.1608085QF12), CCF-Venustech Hongyan Research Initiative (No.CCF-VenustechRP1026002), Suzhou Research Foundation (No. SYG201625), Youth Innovation Promotion Association CAS (No. 2017497), and Fundamental Research Funds for the Central Universities (WK2150110003).

AVAILABILITY

An open source CloudSW (GNU GPL v.2) are freely available at: https://github.com/xubo245/CloudSW.

REFERENCES

[1] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *Journal of molecular biology,* vol. 147, pp. 195-197, 1981.

[2] E. Rucci, C. García, G. Botella, A. De Giusti, M. Naiouf, and M. Prieto-Matías, "State-of-the-Art in Smith–Waterman Protein Database Search on HPC Platforms," in *Big Data Analytics in Genomics*, ed: Springer, 2016, pp. 197-223.

[3] H. Li and R. Durbin, "Fast and accurate short read alignment with Burrows–Wheeler transform," *Bioinformatics,* vol. 25, pp. 1754-1760, 2009.

[4] H. Li and R. Durbin, "Fast and accurate long-read alignment with Burrows–Wheeler transform," *Bioinformatics,* vol. 26, pp. 589-595, 2010.

[5] Y. Liu and B. Schmidt, "Long read alignment based on maximal exact match seeds," *Bioinformatics,* vol. 28, pp. i318-i324, 2012.

[6] B. Langmead and S. L. Salzberg, "Fast gapped-read alignment with Bowtie 2," *Nature methods,* vol. 9, pp. 357-359, 2012.

[7] B. Buchfink, C. Xie, and D. H. Huson, "Fast and sensitive protein alignment using DIAMOND," *Nature methods,* vol. 12, pp. 59-60, 2015.

[8] T. Rognes and E. Seeberg, "Six-fold speed-up of Smith–Waterman sequence database searches using parallel processing on common microprocessors," *Bioinformatics,* vol. 16, pp. 699-706, 2000.

[9] M. Farrar, "Striped Smith‐Waterman speeds database searches six times over other SIMD implementations," *Bioinformatics,* vol. 23, pp. 156-161, 2007.

[10] M. Zhao, W.-P. Lee, E. P. Garrison, and G. T. Marth, "SSW Library: an SIMD Smith-Waterman C/C++ library for use in genomic applications," *PloS one,* vol. 8, p. e82138, 2013.

[11] J. Daily, "Parasail: SIMD C library for global, semi-global, and local pairwise sequence alignments," *BMC bioinformatics,* vol. 17, p. 1, 2016.

[12] P. Muir, S. Li, S. Lou, D. Wang, D. J. Spakowicz, L. Salichos*, et al.*, "The real cost of sequencing: scaling computation to keep pace with data generation," *Genome biology,* vol. 17, p. 1, 2016.

[13] S. Vijayakumar, A. Bhargavi, U. Praseeda, and S. A. Ahamed, "Optimizing sequence alignment in cloud using hadoop and mpp database," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, 2012, pp. 819-827.

[14] G. Zhao, C. Ling, and D. Sun, "SparkSW: scalable distributed computing system for large-scale biological sequence alignment," in *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*, 2015, pp. 845-852.

[15] B. Xu, C. Li, H. Zhuang, J. Wang, Q. Wang, J. Zhou*, et al.* (2017, January 1, 2017). DSA: Scalable Distributed Sequence Alignment System Using SIMD Instructions. *ArXiv e-prints 1701*. Available: https://arxiv.org/abs/1701.01575

[16] O. Gotoh, "An improved algorithm for matching biological sequences," *Journal of molecular biology,* vol. 162, pp. 705-708, 1982.

[17] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer*, et al.*, "The sequence alignment/map format and SAMtools," *Bioinformatics,* vol. 25, pp. 2078-2079, 2009.

[18] X. Feng, H. Jin, R. Zheng, Z. Shao, and L. Zhu, "Implementing Smith-Waterman Algorithm with Two-Dimensional Cache on GPUs," in *Cloud and Green Computing (CGC), 2012 Second International Conference on*, 2012, pp. 25-30.

[19] B. E. Suzek, H. Huang, P. McGarvey, R. Mazumder, and C. H. Wu, "UniRef: comprehensive and non-redundant UniProt reference clusters," *Bioinformatics,* vol. 23, pp. 1282-1288, 2007.

[20] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets," *HotCloud,* vol. 10, pp. 10-10, 2010.

[21] S. Salloum, R. Dautov, X. Chen, P. X. Peng, and J. Z. Huang, "Big data analytics on Apache Spark," *International Journal of Data Science and Analytics,* pp. 1-20, 2016.

[22] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley*, et al.*, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, 2012, pp. 2-2.

[23] H. Li, A. Ghodsi, M. Zaharia, E. Baldeschwieler, S. Shenker, and I. Stoica, "Tachyon: Memory throughput i/o for cluster computing frameworks," *memory,* vol. 18, p. 1, 2013.

[24] H. Li, A. Ghodsi, M. Zaharia, S. Shenker, and I. Stoica, "Reliable, memory speed storage for cluster computing frameworks," *Proc. SoCC,* 2014.

[25] *Aliyun Website*. Available: http://www.aliyun.com

[26] Y. Liu, D. L. Maskell, and B. Schmidt, "CUDASW++: optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units," *BMC research notes,* vol. 2, p. 1, 2009.

[27] E. F. D. O. Sandes, A. Boukerche, and A. C. M. A. D. Melo, "Parallel Optimal Pairwise Biological Sequence Comparison: Algorithms, Platforms, and Classification," *ACM Computing Surveys (CSUR),* vol. 48, p. 63, 2016.

[28] E. Rucci, C. García, G. Botella, A. De Giusti, M. Naiouf, and M. Prieto‐Matías, "An energy‐aware performance analysis of SWIMM: Smith–Waterman implementation on Intel's Multicore and Manycore architectures," *Concurrency and Computation: Practice and Experience,* vol. 27, pp. 5517-5537, 2015.

[29] A. Khalafallah, H. F. Elbabb, O. Mahmoud, and A. Elshamy, "Optimizing smith-waterman algorithm on graphics processing unit," in *Computer Technology and Development (ICCTD), 2010 2nd International Conference on*, 2010, pp. 650-654.

[30] E. Rucci, C. Garcia, G. Botella, A. E. De Giusti, M. Naiouf, and M. Prieto-Matias, "OSWALD: OpenCL Smith–Waterman on Altera's FPGA for Large Protein Databases," *International Journal of High Performance Computing Applications,* p. 1094342016654215, 2016.

[31] Y. Liu and B. Schmidt, "SWAPHI: Smith-Waterman protein database search on Xeon Phi coprocessors," in *2014 IEEE 25th International Conference on Application-Specific Systems, Architectures and Processors*, 2014, pp. 184-185.

[32] H. Lan, W. Liu, B. Schmidt, and B. Wang, "Accelerating large-scale biological database search on Xeon Phi-based neo-heterogeneous architectures," in *Bioinformatics and Biomedicine (BIBM), 2015 IEEE International Conference on*, 2015, pp. 503-510.

[33] Y. Liu, A. Wirawan, and B. Schmidt, "CUDASW++ 3.0: accelerating Smith-Waterman protein database search by coupling CPU and GPU SIMD instructions," *BMC Bioinformatics,* vol. 14, p.: 117., 2013.

[34] X. Meng and V. Chaudhary, "A high-performance heterogeneous computing platform for biological sequence analysis," *IEEE Transactions on Parallel and Distributed Systems,* vol. 21, pp. 1267-1280, 2010.

[35] E. Rucci, A. De Giusti, M. Naiouf, G. Botella, C. García, and M. Prieto-Matias, "Smith-Waterman algorithm on heterogeneous systems: A case study," in *2014 IEEE International Conference on Cluster Computing (CLUSTER)*, 2014, pp. 323-330.

[36] S. Gálvez, A. Ferusic, F. J. Esteban, P. Hernández, J. A. Caballero, and G. Dorado, "Speeding-up Bioinformatics Algorithms with Heterogeneous Architectures: Highly Heterogeneous Smith-Waterman (HHeterSW)," *Journal of Computational Biology,* 2016.