SeedHit: a GPU friendly pre-align filtering algorithm

Zhen Ju, Jingjing Zhang, Xuelei Li, Jintao Meng, Yanjie Wei

***Abstract*—The amount of genetic data generated by Next Generation Sequencing (NGS) technologies grows faster than Moore's law. This necessitates the development of efficient NGS data processing and analysis algorithms. A filter before the computationally-costly analysis step can significantly reduce the run time of the NGS data analysis. As GPUs are orders of magnitude more powerful than CPUs, this paper proposes a GPU-friendly pre-align filtering algorithm named SeedHit for the fast processing of NGS data. Inspired by BLAST, SeedHit counts seed hits between two sequences to determine their similarity. In SeedHit, a nucleic acid in a gene sequence is presented in binary format. By packaging data and generating a lookup table that fits into the L1 cache, SeedHit is GPU-friendly and high-throughput. Using three 16s rRNA datasets from Greengenes as input SeedHit can reject 84%-89% dissimilar sequence pairs on average when the similarity is 0.9-0.99. The throughput of SeedHit achieved 1 T/s (Tera base per second) on 3080 Ti. Compared with the other two GPU-based filtering algorithms, GateKeeper and SneakySnake, SeedHit has the highest rejection rate and throughput. By incorporating SeedHit into our in-house clustering algorithm nGIA, the modified nGIA achieved a 1.6-2.1 times speedup compared to the original version.**

***Index Terms*—Pre-align filter, seed hit, GPU.**

I. Introduction

The rapid development of gene sequencing technology has resulted in an explosion of genetic sequence data. For instance, The 1000 Genomes Project [1], [2] generated over 50 TB of data. A more ambitious project is the Earth BioGenome Project [3], which aims to sequence all known eukaryotic species and generate massive genetic sequence data. As a result, bioinformatics algorithms and pipelines for gene annotation, evolutionary analysis, sequence clustering, and other tasks face increasing demands for memory and computing power.

Among all bioinformatics algorithms for gene sequence analysis, clustering is an essential process [4]. Holm proposed a greedy incremental sequence clustering method [5], which uses an alignment step to determine whether two sequences are similar. The alignment is a dynamic programming process [6], and the acceleration of the alignment can dramatically increase the efficiency of the clustering algorithm. Many strategies have been proposed to perform and accelerate sequence alignment on different platforms such as CPU [7], [8], FPGA [9], [10], and GPU [11], [12]. On the other hand, a pre-alignment filter can reduce the sequences for alignment and thus can be very effective for clustering. An efficient filter can be introduced to speed up the alignment step in bioinformatics pipelines [5].

In this paper, we proposed an efficient and high-throughput pre-align filtering algorithm called SeedHit to accelerate the sequence clustering tool without reducing the precision of the clustering results. The SeedHit filtering algorithm is an efficient, high-throughput, pre-alignment filter developed on GPU. GPUs have a large number of cores optimized for parallel processing, and larger cache sizes compared to CPUs. Each thread on a GPU is allocated a smaller portion of the overall cache, allowing for efficient data access and processing across multiple threads. This design enables GPUs to handle a large number of threads in parallel, making them well-suited for compute-intensive applications. As in BLAST, a pair of similar sequences sharing enough short sequences are called seed hit in our paper. The key idea is to count the seed hits to determine whether the two sequences will be similar. The SeedHit algorithm packages data to improve bandwidth utilization and builds a lookup table that can be put into the cache to avoid random memory access.

The contributions of this paper are as follows: (1) This paper proposed an efficient pre-align filtering algorithm, which can guarantee a high rejection rate without introducing errors; (2) This paper implemented the SeedHit filtering algorithm on the GPU to improve the high throughput of the filter; (3) Compared with other state-of-the-art GPU-based pre-align filters like GateKeeper and SneakySnake, SeedHit achieves the highest throughput on different GPUs and datasets; (4) The SeedHit filtering algorithm is applied to our in-house gene clustering method, nGIA, and the modified version runs 1.6x faster than the original nGIA.

II. Related work

Early filtering algorithms assess sequence similarity by quantifying shared subsequences in a sequence pair. CD-HIT proposed a short word filtering algorithm, that gauges similarity based on shared short word counts in sequence pairs [13]. Although offering both high accuracy and low computation, the short word filter wasn't optimized for throughput.

The Shifted Hamming Distance (SHD) algorithm employs edit distance between a sequence pair to assess sequence similarity [14]. SHD relies on SIMD instructions such as SSE, experiencing significant performance degradation on other platforms like GPU. DiagAF algorithm is based on SHD and can filter unequal-length sequences, reducing computational complexity by focusing solely on elements near the diagonal [15]. Due to the limited computation power of CPUs, DiagAF's throughput remains within the same order of magnitude as SHD. RattlesnakeJake also utilizes edit distance for filtering, utilizing memristors for in-memory computations to reduce data movement overheads [16]. While RattlesnakeJake

only supports short sequence filtering, an extended version for long sequence filtering is developed based on the memristors platform [17].

GateKeeper is a filtering algorithm inspired by the pigeonhole principle and implemented on an FPGA platform, far surpassing SHD-type algorithms in throughput [18]. Later both a faster FPGA based GateKeeper named Shouji and a faster GPU based Gatekeeper are developed [19], [20].

The Bit Matrix Algorithm, similar to Shouji, runs on CPU platforms and employs multi-threading for acceleration [21]. Limited by the CPU's computational power, the throughput of the Bit Matrix Algorithm remains within the same order of magnitude as Shouji. SneakySnake, implementing a Shouji-like filtering algorithm on GPU platforms, outperforms GateKeeper and Shouji significantly [22]. GPU based GateKeeper and SneakySnake are currently the fastest filtering algorithms, which are compared with our proposed SeedHit in this paper.

III. Methods

This paper proposed a pre-align filtering algorithm SeedHit, which can reject different sequence pairs efficiently. SeedHit is developed based on GPU for acceleration. The key idea of the SeedHit filter is inspired by BLAST [23], which finds similar sequences by locating and extending seeds between the two sequences. The similarity of sequence pairs is the ratio of the length of the LCS(longest common subsequence) to that of the shorter sequence. The k-mer is a substring of length k contained within a biological sequence. If two sequences are similar, there must be enough k-mers in a sequence that can match another sequence, and the matched k-mers are called seed hits. Also, the idea of seed hits is employed by several sequence alignment algorithms [23], [24]. SeedHit first cuts the sequence to k-mers, then compare each k-mer to another sequence to count the seed hits. If there are enough seed hits, the filter accepts the two sequences; otherwise, it rejects them.

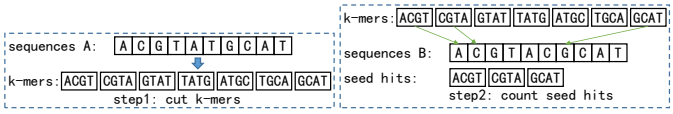


Fig. 1. Key idea of SeedHit filter.

A. Overview of SeedHit algorithm

For algorithms like SeedHit, the longer the k-mer, the better the filtering performance. Due to the limitation of lookup table size and similarity of sequences, the length of k-mer ranges from 4 to 8 (integer). A gene sequence is composed of nucleic acids and stored in binary code in the computer. The binary format can describe the original text representation of a k-mer as n-bit-mer (for short, b-mer). The k in k-mer is the length of the k-mer, limited by GPU cache size, ranging from 4 to 8. Each base is represented by two bits, and the length of bits corresponding to k-mers is denoted as b in a b-mer ranging from 8-16. If a nucleic acid is represented and stored in n bits (for example, A, G, C, T as 00,10,01,11 and n=2), then a k-mer (k=5) in the text representation can be described as a b-mer (b=10) in the binary representation. Since the longer the k-mer, the smaller the probability of hitting on another sequence, the longer the k-mer can improve the rejection rate of the filter. SeedHit uses b-mer instead of k-mer for it is longer and allows more fine-grained filtering than the original k-mer representation. The algorithm represented by b-mer in this paper is named SeedHit, and the original algorithm using k-mers is named SeedHit-kmer.

To take full advantage of the computation power of GPU, SeedHit introduced the entropy bits for better data packing and built a lookup table to make seed hits query more efficient. The overall workflow of SeedHit can be divided into the following three steps, as shown in Figure 2.

(1) Data packing: Using the entropy bits to store nucleic acid reduces the communication bandwidth between the host and the device.

(2) Building lookup table: The lookup table records which b-mer can match the representative sequence.

(3) Counting seed hits: For a given sequence, cut b-mers and search them on the lookup table to count seed hits.

Details of the algorithms are described in the following several subsections.

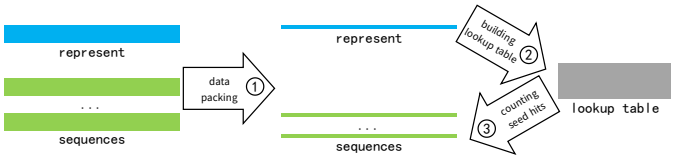


Fig. 2. Schematic diagram of SeedHit algorithm

B. Data packing

SeedHit packs each nucleic acid of A/G/C/T into 2 bits, while each nucleic acid in SeedHit with k-mer takes 8 bits. SeedHit divides the 2 bits of nucleic acid into two different INTs, as shown in Figure 3 to save the memory and allow bitwise operation.

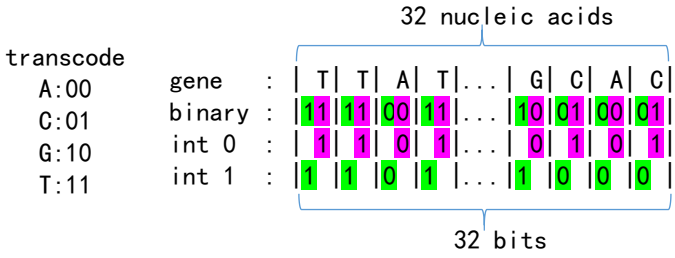


Fig. 3. Data packing of a gene sequence.

Figure 3 shows the data packing process of a gene sequence with a length of 32. On the left is the transcode table, which converts A, C, G, and T to 00, 01, 10, and 11, respectively. On the right shows the process of data packing. The first row denotes the original sequence, and the second row is the binary format converted from the original sequence according to the transcode table. The third and fourth rows are the results that store each base's high and low bits in the binary format.

C. Building lookup table

It is expensive to query whether a b-mer can match the representative sequence. SeedHit records all possible b-mers and whether they can match the representative sequence by a lookup table, as shown in Figure 4. Any b-mer can be quickly checked by querying the lookup table.

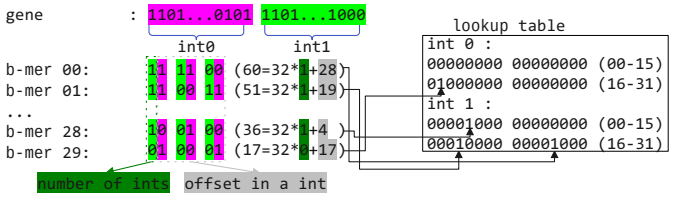


Fig. 4. Building the lookup table. The length of the sequences is 32 and the b-mer is 6.

To calculate the position of a b-mer in the lookup table through shift operation, the rightmost 5 bits of a b-mer determine its offset in an INT, while the left bits determine the INT in which it is stored. The lookup table contains all possible b-mers, and each b-mer must be queried, resulting in random memory accesses. To avoid these random accesses, SeedHit stores the lookup table in the L1 cache, limiting its size to 32KB.

D. Filtering by seed hits

The number of seed hits can be used to determine the similarity between two sequences. Given a binary sequence of length l and a similarity threshold of s, there can be at most $(1.0 - s) * l$ dissimilar bits between the sequences. If the length of a b-mer is k, the sequence can be divided into $l - k + 1$ b-mers. At most, $(1.0 - s) * l * k$ b-mers will be polluted by dissimilar bits. Therefore, for a given sequence, a similar sequence must have at least $l-k+1-(1.0-s)*l*k$ seed hits. Considering that each base has two bits and the b-mer should occur every two bits, the calculation formula is $(l-k)/2+1-(1.0-s)*(l/2)*(k/2)$. If a sequence has enough seed hits, it will be accepted, otherwise it will be filtered out, as shown in Algorithm 1.

Since the filtering process of each sequence is independent of the other, it is ideal to use GPU for acceleration. Keeping the lookup table in the GPU's shared memory, each GPU thread processes a pair of sequences as shown in Algorithm 1.

Algorithm 1: Counting seed hits

Input: packed binary sequence p
Input: lookup table of represent sequence $table$
Output: number of seed hits c
// parallel on stream processor
unsigned int $nbit$;
for *get 32 bases from* p **do**
  low 32 bit $int0$;
  high 32 bit $int1$;
  for $i \leftarrow 31$**to**0 **do**
    $nbit = nbit + (int0 >> i\&1)$;
    $nbit = nbit + (int1 >> i\&1) << 1$;
    $nbit = nbit << 2$;
    // number of ints
    unsigned int $number = nbit >> 5$;
    // offset in a int
    unsigned int $offset = nbit\&0B11111$;
    // number of seed hits
    $c+ = table[unmber]\&(1 << offset) >> offset$;
  end
end

TABLE I
ILLUSTRATION OF SEQUENCE FILTERING

Sequence pair	seqA: ACAGTGCACT seqB: ACATCTTACA	seqA: CGCTACGCTA seqB: CGCTGGCTA
Data Packing	seqA: 0001001011100100111 seqB: 0010001101111110010	seqA: 0110011100011001110 seqB: 0111011010111110
b-mers in SeqA	00010 01001 00101 10111 11100 10010 01000 00011	011001 100111 011100 110001 000110 011001 100111 011100
Building lookup table	00010 01001 00101 10111 11100 10010 01000 00011	011001 100111 011100 110001 000110
b-mers in seqB	00010 01001 00110 11011 01111 11110 11000 00010	011001 100111 011110 111010 101001 100111 011100
seed hits	00010 01001 00010	011001 100111 100111 011100
seed hits count	3	4
seed hits needed	6	3.6
Filtering by seed hits	no	yes

E. Illustration of sequence filtering

Two pairs of sequences are taken for examples to better illustrate the SeedHit algorithm. The similarity cutoff for both pairs of sequences is 0.9. The length of the sequences in the first pair is 10 with a b-mer of 5, while the second pair has lengths of 10 and 8 respectively, with a b-mer of 6. Table I displays the outcomes of "data packaging," "building lookup table," and "Filtering by seed hits" for these sequence pairs.

Table I demonstrates the filtering process for two sequence pairs. The "Data packing" step packages the sequences into binary. Select the longer sequence from a pair as seqA and the other as seqB, then convert them into binary format. The "Building lookup table" step constructs a lookup table based on seqA's b-mer by extracting substrings of length b from binary seqA every 2 bits as b-mers. The lookup table consists of unique b-mers. In the "Filtering by seed hits" step, the shared b-mer count determines if the sequence pair meets the criteria. Apply the same method used for seqA to segment seqB into b-mers. Calculate the occurrences of b-mers found in the lookup table, resulting in 3 for the first pair and 4 for the second pair. The minimum seed hits required are 6 for the first pair and 3.6 for the second pair (as per the formula in the preceding section). The first pair did not meet the conditions, while the second pair passed.

IV. Results and Discussion

This paper compares SeedHit with two established filtering algorithms, GateKeeper and SneakySnake, by conducting tests on three datasets and three GPU cards. The evaluation experiments include the following details: (i) a comparison of the performance (recall, precision and F1 score) of SeedHit, SneakySnake, and GateKeeper on three 16S rRNA datasets; (ii) a comparison of the throughput of SeedHit, SneakySnake, and GateKeeper on three distinct GPUs; (iii) an analysis of the impact of the SeedHit filter on nGIA, a sequence clustering tool developed by our team.

A. Datasets and GPUs

In this paper, we assess the proposed algorithms by utilizing three widely recognized 16S rRNA datasets, namely NCBI, SILVA, and RDP, which have sequence lengths ranging from 1250-1672. These datasets were obtained from GREENGENES (https://GREENGENES.lbl.gov) [25]. To provide a comprehensive evaluation of the algorithms, we generated seven new datasets. DS1, DS2, and DS3 datasets were created by extracting the first 128 bases from NCBI, SILVA, and RDP, respectively. DS4, DS5, DS6, and DS7 datasets were generated by combining NCBI, SILVA, and RDP and selecting only the first 128, 256, 512, and 1024 bases of each sequence, respectively. The specifics, including the number and length ranges of sequences, for each dataset are presented in Table 1.

TABLE II
Datasets used for evaluation

Dataset	NCBI	SILVA	RDP	DS1	DS2
Count	97,413	381,226	711,278	97,413	381,226
Length	1250-1570	1251-1672	1251-1672	128	128
Dataset	DS3	DS4	DS5	DS6	DS7
Count	711,278	2,379,806	2,379,806	2,379,806	2,379,806
Length	128	128	256	512	1024

To compare the performance of SeedHit with SneakySnake and GateKeeper, this paper selects Nvidia 1080 Ti, 2080 Ti, and 3080 Ti, which have Pascal, Turing, and Ampere architectures, respectively. The 1080 Ti, 2080 Ti, and 3080 Ti GPU has 48KB, 64KB, and 128KB L1 cache, respectively. All the three algorithms uses Nvidia's CUDA programming framework. The number of CUDA core for three GPUs are 3584, 4352, and 10240, respectively.

B. Performance comparison

This paper evaluates the accuracy of filtering algorithms using three metrics: recall, precision, and F1 score [26]. As filtering algorithms decide whether a sequence pair is accepted or rejected, it can be considered a binary classification algorithm. The formulas for recall, precision, and F1 score are as follows: $precision = TP/(TP + FP), recall = TP/(TP + FN), F1\ score = 2{*}precision{*}recall/(precision{+}recall)$. Here, TP, FP, and FN denote the numbers of correctly rejected, incorrectly rejected, and incorrectly accepted sequence pairs by the filter, respectively.

In this section, we will assess the recall of SeedHit on the NCBI, SILVA, and RDP datasets. The recall metric measures the filter's ability to decrease the requirement for computationally-intensive alignment procedures. To compare the performance of SeedHit with other cutting-edge GPU-based pre-alignment filters, we will also evaluate the recall of SneakySnake and GateKeeper.

The GPU-based SneakySnake tool supports a maximum sequence length of 128, while SeedHit can handle up to 65536. Therefore, we utilized DS1, DS2, and DS3 datasets with sequence lengths of 128 for comparison. SeedHit requires selecting the length of b-mers. The method involves filtering the first 100 sequences, iterating through b-mer lengths ranging from 8 to 16, and selecting the b-mer that rejects the most sequence pairs. Due to the small number of sequences, the time consumption can be ignored. To filter the data, we selected one representative sequence for every 1000 sequences and compared all remaining sequences to the representative sequence. In total, we performed 97, 381, and 711 runs of filtering for DS1, DS2, and DS3 datasets, respectively. We calculated the average recall over these runs and present the results in Figure 5.

Figure 5 displays the results, where the X-axis represents similarity, and the Y-axis represents recall. The blue, orange, gray, and yellow curves correspond to the recall of SeedHit, SeedHit with k-mer, SneakySnake, and GateKeeper, respectively. Subgraphs (a), (b), and (c) illustrate the outcomes for datasets DS1, DS2, and DS3.

SeedHit utilizes b-mer instead of k-mer to enhance efficiency. As depicted in Figure 5(a), when the similarity is 0.9, SeedHit's recall is 2.18% higher than that of SeedHit with k-mer. SeedHit consistently outperforms SeedHit with k-mer in recall for similarity values ranging from 0.9 to 0.99, although the difference narrows as the similarity increases. The trends observed in Figure 5(b) and (c) are similar to those in Figure 5(a).

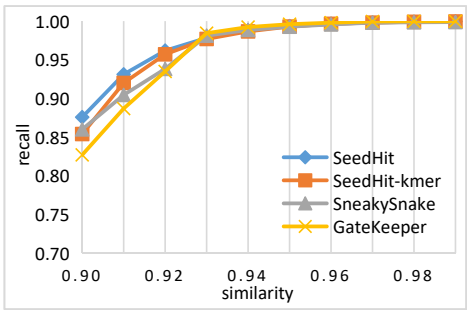
SeedHit is a highly efficient filtering algorithm, with average recalls of 97.21%, 97.04%, and 96.93% on DS1, DS2, and DS3, respectively. Figure 5(a) compares the recall of SeedHit, SneakySnake, and GateKeeper on DS1, with SeedHit achieving the highest recall among the three algorithms for all similarity values and an average recall that is 0.7% higher than the second-best algorithm. Figure 5(b) exhibits a similar trend to Figure 5(a), with SeedHit's average recall being 0.9% higher than the second-best algorithm. In Figure 5(c), SeedHit's recall ranks second, with an average recall that is 0.6% lower than the top-performing algorithm.

Another indicator of the binary classification problem is precision, which can measure the accuracy of a filter. The corresponding precision of the experiments in Figure 5 is shown in Figure 6.

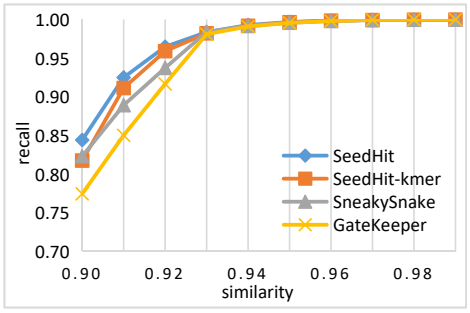
Figure 6 displays precision on the Y-axis, with the same parameters as Figure 5. The graph illustrates that the precision of SneakySnake increases as the similarity value increases, while the precision of the other filters remains at 100%.

SeedHit is a highly effective and accurate filtering algorithm. As depicted in Figure 5, SeedHit and SneakySnake exhibit recall rates of up to 97%, making them efficient

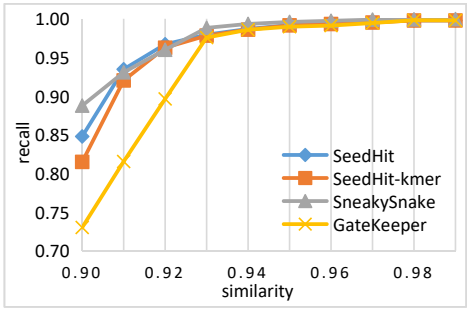
This article has been accepted for publication in IEEE/ACM Transactions on Computational Biology and Bioinformatics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCBB.2024.3417517



(a) DS1

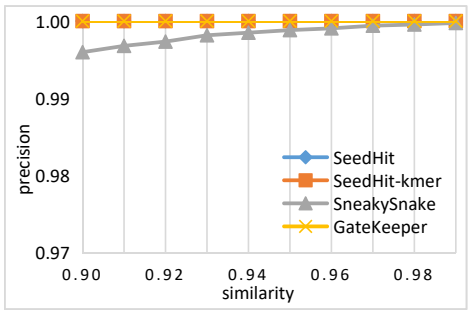


(b) DS2

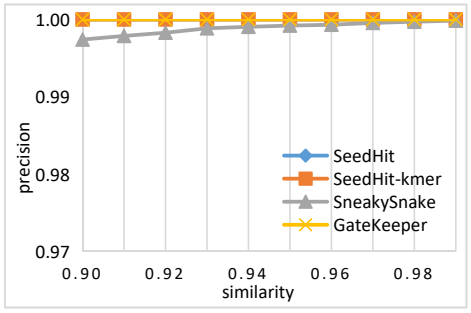


(c) DS3

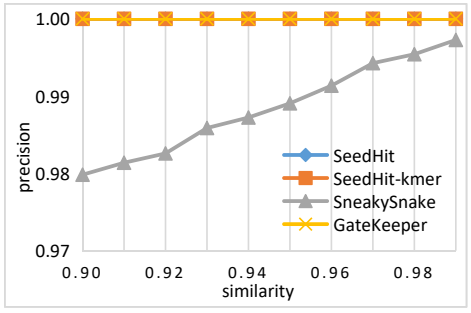
Fig. 5. The recall of SeedHit, SneakySnake, and GateKeeper.



(a) DS1



(b) DS2



(c) DS3

Fig. 6. The precision of SeedHit, SneakySnake, and GateKeeper.

filtering algorithms. Similarly, Figure 6 illustrates that SeedHit and GateKeeper demonstrate precision rates of up to 100%, making them accurate filtering algorithms. While SneakySnake has a recall rate comparable to SeedHit, its precision rate is lower. Conversely, while GateKeeper's precision rate is on par with SeedHit, its recall rate is lower.

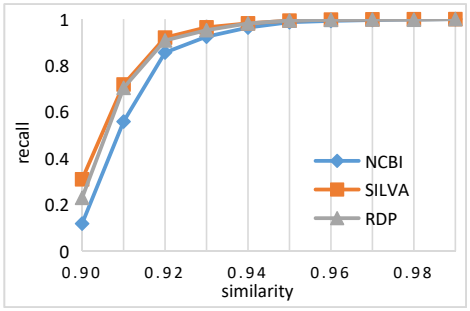
SeedHit is capable of filtering sequence pairs with different lengths, whereas SneakySnake and GateKeeper do not support this feature. To assess SeedHit's performance on datasets with varying lengths, we conducted additional tests on NCBI, SILVA, and RDP datasets using the same experimental method as DS1, DS2, and DS3, but without cutting sequences. The results are presented in Figure 7. In Figure 7, the X-axis represents similarity, while the Y-axis in subgraphs (a) and (b) represent recall and precision, respectively. The blue, orange, and gray curves correspond to the NCBI, SILVA, and RDP datasets.

As in Figure 7, as the similarity increases, the recall also increases. The number of seed hits required is influenced by the similarity and the length gap between sequence pairs. In Figure 5 and Figure 7, the recall of SeedHit increased with the similarity because more seed hits are required for greater similarity value. However, the recall in Figure 7 is lower than that in Figure 5 due to the presence of sequence pairs with different lengths, which reduces the number of required seed hits.

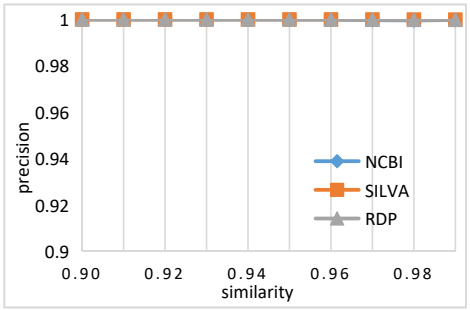
The F1 score serves as a comprehensive assessment parameter. This study employs DS1, DS2, and DS3 as input, with similarity ranging from 0.9 to 0.99 at intervals of 0.01. The SeedHit, SeedHit-kmer, SneakySnake, and GateKeeper algorithms are tested on each dataset. For each algorithm and dataset, 10 F1 scores are computed across varying similarities. The average of these F1 scores is plotted as Figure 8.

Figure 8 shows that the SeedHit-kmer algorithm, utilizing k-mers, demonstrates a superior F1 score compared to SneakySnake and GateKeeper on DS1 and DS2, and a comparable performance to SneakySnake on DS3. Meanwhile, the SeedHit algorithm with b-mer representation consistently achieves the highest F1 score across all three datasets.

This article has been accepted for publication in IEEE/ACM Transactions on Computational Biology and Bioinformatics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCBB.2024.3417517



(a) recall of SeedHit



(b) precision of SeedHit

Fig. 7. The recall and precision of SeedHit on NCBI, SILVA, and RDP.

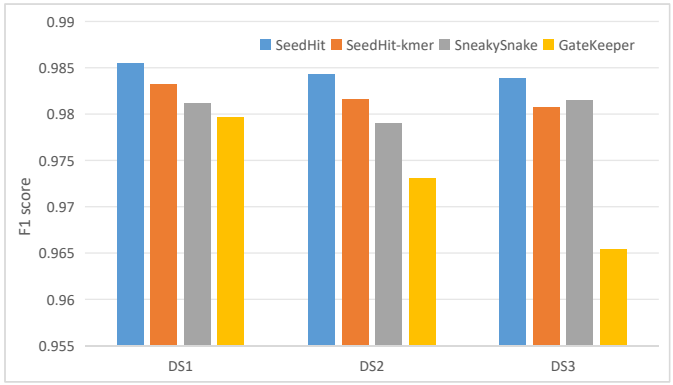
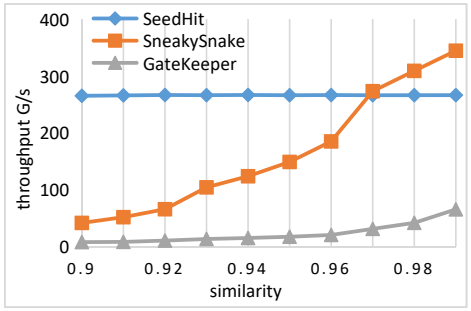
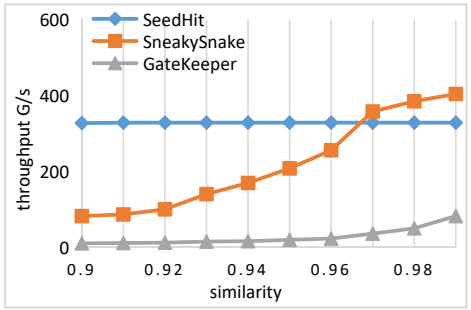


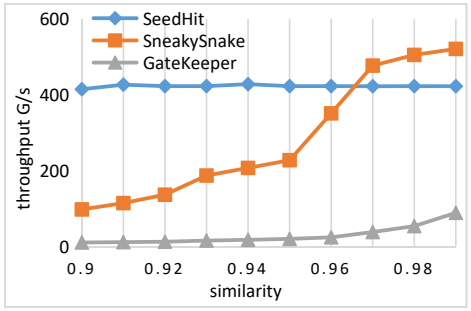
Fig. 8. F1 score of SeedHit, SeedHit-kmer, SneakySnake, and GateKeeper.



(a) DS1



(b) DS2



(c) DS3

Fig. 9. Throughput of SeedHit, SneakySnake, and GateKeeper on 1080 Ti with DS1, DS2, and DS3.

C. *Throughput of algorithms*

This section focuses on evaluating the throughput of the kernel function of the algorithms. Throughput refers to the number of bases processed by a filter's kernel function within a specific time period. We evaluated the throughput of SeedHit, SneakySnake, and GateKeeper on DS1, DS2, and DS3 datasets. To obtain the results, we selected the first sequence of each dataset as the representative and filtered it with all other sequences. We repeated this process 100 times and calculated the average.

Figure 9 displays the results on Nvidia 1080 Ti. The X-axis represents similarity, while the Y-axis represents throughput. The blue, orange, and gray curves correspond to the throughput of SeedHit, SneakySnake, and GateKeeper, respectively. Subgraphs (a), (b), and (c) show the results of the DS1, DS2, and DS3 datasets.

The throughput of SeedHit on DS1 is consistently high and robust. As shown in Figure 9(a), the throughput of SneakySnake and GateKeeper changes as the similarity decreases, while SeedHit maintains a steady throughput. The time complexity of SneakySnake and GateKeeper is proportional to the 'Error Threshold', which is calculated as '$sequence\ length * (1.0 - similarity)$'. Decreasing similarity or increasing sequence length increases the 'Error Threshold', resulting in reduced throughput. SneakySnake has the highest throughput when the similarity is 0.99, 0.98, and 0.97, while SeedHit has the highest throughput in all other cases. The average throughput of SeedHit over similarity values 0.9 to 0.99 is 61% and 1037% higher than that of the SneakySnake and GateKeeper. The similar trend is observed for DS2 and DS3, with SeedHit having an about 50% and 1107%-1343% higher throughput than the SneakySnake and GateKeeper. GPUs are designed to handle a large number of parallel tasks, and since DS3 has more sequences, it can increase the utilization of the GPU.

We also evaluate the impact of different b-mer lengths on

This article has been accepted for publication in IEEE/ACM Transactions on Computational Biology and Bioinformatics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCBB.2024.3417517

throughput for different GPUs. As in Figure 9, the similarity has no effect on the throughput of SeedHit, thus in this evaluation the similarity is uniformly set to 0.9. Using DS1, DS2, and DS3 datasets, we set b-mer lengths ranging from 8 to 16, and the results are shown in Figure 10. Figure 10 shows that the throughput of SeedHit varies slightly with the changing b-mer. The throughput increases with the growth of the dataset size and the GPU's computing capability.

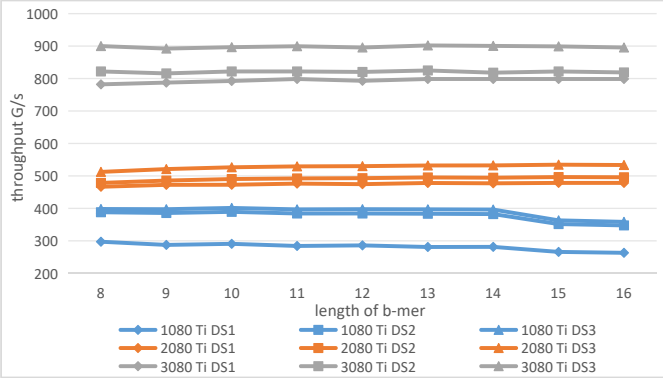


Fig. 10. Throughput of SeedHit on 1080 Ti, 2080 Ti, and 3080 Ti, with DS1, DS2, and DS3.

In this paper, we further computed the throughput of SeedHit, SneakySnake, and GateKeeper on 1080 Ti, 2080 Ti, and 3080 Ti GPUs. The results are presented in Table III. The

TABLE III
SEEDHIT, SNEAKYSNAKE AND GATEKEEPER'S THROUGHPUT OF DS1, DS2, AND DS3 DATASETS ON 1080 TI, 2080 TI, AND 3080 TI.

	DS1	DS2	DS3
		1080 Ti	
SeedHit (G/s)	**266**	**328**	**423**
SneakySnake (G/s)	165	218	282
GateKeeper (G/s)	23	27	29
		2080 Ti	
SeedHit (G/s)	**435**	**538**	**570**
SneakySnake (G/s)	303	472	534
GateKeeper (G/s)	31	45	50
		3080 Ti	
SeedHit (G/s)	**630**	**823**	**885**
SneakySnake (G/s)	453	700	760
GateKeeper (G/s)	42	69	81

reported throughput is the average of each filter's throughput over similarities ranging from 0.9 to 0.99. The best result of each test case is achieved by SeedHit (shown in bold). For a given GPU card, the throughput of all filters increases with the increasing size of Dataset, though the increase of GateKeeper is marginal. Across different platforms and datasets, SeedHit performed best as shown in Table III. Specifically, on the 2080 Ti GPU, SeedHit's throughputs in the three datasets are 44%, 14%, and 7% higher than the SneakySnake (second-best), while on the 3080 Ti, the throughputs are 39%, 18% and 16% higher than the SneakySnake.

The SeedHit algorithm demonstrates robust performance across GPUs of varied architectures. The 1080 Ti, 2080 Ti, and 3080 Ti represent the Pascal, Turing, and Ampere architectures, respectively. SeedHit's throughput on these three generations of GPUs also shows an increasing trend, surpassing that of other algorithms on any given GPU. This underscores SeedHit's adaptability to diverse GPU architectures.

To further evaluate the maximum throughput of SeedHit, we used the DS4, DS5, DS6, and DS7 datasets to maximize GPU utilization. These datasets have varying sequence lengths, allowing us to evaluate SeedHit's throughput with longer sequences. Table IV presents SeedHit's throughput on the DS4, DS5, DS6, and DS7 datasets using 1080 Ti, 2080 Ti, and 3080 Ti GPUs. The reported throughput is the average of each filter's throughput with similarities ranging from 0.9 to 0.99.

TABLE IV
SEEDHIT'S THROUGHPUT OF DS4, DS5, DS6, AND DS7 DATASETS ON 1080 TI, 2080 TI, AND 3080 TI.

	DS4	DS5	DS6	DS7
1080 Ti (G/s)	428	440	429	423
2080 Ti (G/s)	581	641	639	633
3080 Ti (G/s)	913	1,030	956	904

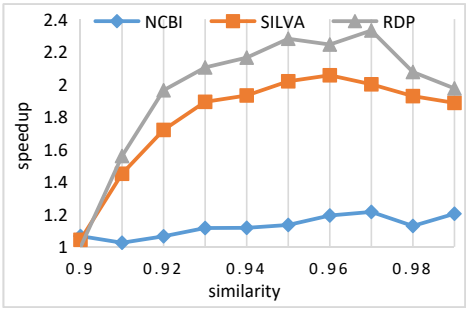
The results in Table IV show that SeedHit's throughput on 1080 Ti varies between 423-440 G/s for variable sequence lengths. Dividing the difference between the maximum and minimum values by the minimum value yields a throughput variation of 4% for 1080 Ti. However, on 2080 Ti and 3080 Ti, the variation of SeedHit's throughput for variable sequence lengths is about 9% and 12%, respectively. The maximum throughput for SeedHit is achieved when the sequence length is 256 (DS5). The performance of the SeedHit algorithm exhibits a strong correlation with GPU's memory bandwidth. A sequence length of 256 aligns optimally with retrieving data from graphics memory in a single cycle. Deviations from this optimal length pose inefficiencies: sequences shorter than 256 necessitate padding to this length resulting in bandwidth waste, while longer sequences require multiple reads. Compared to SneakySnake and GateKeeper, SeedHit has a higher throughput and can reach 1T bases/s on 3080 Ti.

D. Impact on sequence clustering

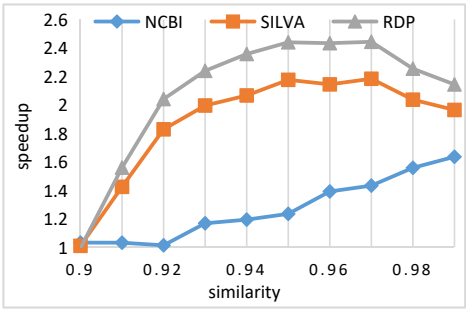
To assess the impact of SeedHit on bioinformatics applications, we integrated SeedHit into a clustering tool called nGIA [27]. In this section, we compare the end-to-end performance of nGIA with and without the SeedHit filter by calculating the speedup. We ran both the modified and original versions of nGIA on 1080 Ti, 2080 Ti, and 3080 Ti GPUs using NCBI, SILVA, and RDP as input. The results are presented in Figure 11.

Figure 11 illustrates the speedup achieved by the modified nGIA with SeedHit. The X-axis represents similarity, while the Y-axis represents speedup. The blue, orange, and gray curves correspond to NCBI, SILVA, and RDP datasets, respectively. Subgraphs (a), (b), and (c) display the results obtained on 1080 Ti, 2080 Ti, and 3080 Ti, respectively.

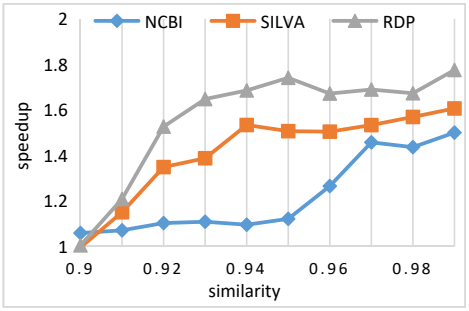
The SeedHit filter has improved the speed of nGIA. As shown in Figure 11(a), the speedup increases with the similarity. This is because as the similarity increases, the recall of SeedHit also increases, reducing the need for the alignment step. When the similarity exceeds 0.95, the rejection rate of



(a) 1080 Ti



(b) 2080 Ti



(c) 3080 Ti

Fig. 11. Speedup of nGIA with SeedHit on 1080 Ti, 2080 Ti, and 3080 Ti.

SeedHit approaches 99%, and further improvement in speed is unlikely. When the NCBI, SILVA, and RDP datasets are used as input, the average speedup of nGIA from similarity 0.9 to 0.99 is 1.1, 1.8, and 2.0, respectively. Speedup is more pronounced for bigger dataset. Figure 11(b) and (c) show the same trend as Figure 11(a), with average speedups of 1.3, 1.9, 2.1, and 1.2, 1.4, 1.6, respectively. Overall the improvement of the SeedHit on the clustering tool nGIA is more pronounced for bigger datasets. For instance, for all three GPU cards, the modified nGIA shows the highest speedup (2.0x, 2.1x, and 1.6x) for the RDP dataset. This indicates that SeedHit can assist other bioinformatics algorithms, such as clustering, in processing large volumes of genetic data.

V. Conclusion

In this paper, we proposed SeedHit, a highly efficient and GPU-based pre-alignment filtering algorithm. Our experiments on various GPUs and datasets demonstrate that the SeedHit performed best compared with the GateKeeper and SneakySnake. Specifically, the average throughput of the SeedHit is about 7% to 61% higher than that of the SneakySnake, and about 998% to 1402% higher than that of GateKeeper. Furthermore, SeedHit can be integrated with sequence clustering tools to accelerate their performance. By incorporating SeedHit into nGIA, the modified nGIA achieved a 1.6-2.1 times speedup compared to the original version. Although SeedHit delivers high performance on different GPUs, its performance improvement is not linearly related to the GPU's memory bandwidth or FP32 performance. This suggests that there is still room for optimization. One possible direction is to optimize the data structure of SeedHit and improve data locality. The SeedHit algorithm can also be used to accelerate other clustering tools in the future.

Acknowledgment

This work was partly supported by the Key Research and Development Project of Guangdong Province under grant no.2021B0101310002, National Key Research and Development Program of China Grant No. 2021YFF1200104, Strategic Priority CAS Project XDB38050100, National Science Foundation of China under grant no.62272449, the Shenzhen Basic Research Fund under grant no RCYX20200714114734194, KQTD20200820113106007 and Shenzhen Key Laboratory of Intelligent Bioinformatics under grant no ZDSYS20220422103800001. We would also like to thank the funding support by the Youth Innovation Promotion Association(Y2021101), CAS to Yanjie Wei.

References

[1] Eric S. Lander, David Altshuler, Mark J. Daly, Sharon Rachel Grossman, and Joshua M. Korn. A map of human genome variation from population-scale sequencing. *Nature*, 457, 2012.

[2] D. M. Altshuler, R. M. Durbin, G. R. Abecasis, D. R. Bentley, and Genomes Project Consortium. An integrated map of genetic variation from 1,092 human genomes consortium gp nature 2012 491 56 65 10.1038/nature11632. *Nature*, 491(7422):56–65, 2012.

[3] Harris A Lewin, Stephen Richards, Erez Lieberman Aiden, Miguel L Allende, John M Archibald, Miklós Bálint, Katharine B Barker, Bridget Baumgartner, Katherine Belov, Giorgio Bertorelle, et al. The earth biogenome project 2020: Starting the clock, 2022.

[4] Quan Zou, Gang Lin, Xingpeng Jiang, Xiangrong Liu, and Xiangxiang Zeng. Sequence clustering in bioinformatics: an empirical study. *Briefings in bioinformatics*, 21(1):1–10, 2020.

[5] Liisa Holm and Chris Sander. Removing near-neighbour redundancy from large protein sequence collections. *Bioinformatics (Oxford, England)*, 14(5):423–429, 1998.

[6] Temple F Smith, Michael S Waterman, et al. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, 1981.

[7] Torbjørn Rognes. Faster smith-waterman database searches with inter-sequence simd parallelisation. *BMC bioinformatics*, 12(1):1–11, 2011.

[8] Mengyao Zhao, Wan-Ping Lee, Erik P Garrison, and Gabor T Marth. Ssw library: an simd smith-waterman c/c++ library for use in genomic applications. *PloS one*, 8(12):e82138, 2013.

[9] Isaac TS Li, Warren Shum, and Kevin Truong. 160-fold acceleration of the smith-waterman algorithm using a field programmable gate array (fpga). *BMC bioinformatics*, 8:1–7, 2007.

[10] Enzo Rucci, Carlos Garcia, Guillermo Botella, Armando De Giusti, Marcelo Naiouf, and Manuel Prieto-Matias. Swifold: Smith-waterman implementation on fpga with opencl for long dna sequences. *BMC systems biology*, 12(5):43–53, 2018.

[11] Yongchao Liu, Adrianto Wirawan, and Bertil Schmidt. Cudasw++ 3.0: accelerating smith-waterman protein database search by coupling cpu and gpu simd instructions. *BMC bioinformatics*, 14:1–10, 2013.

[12] Nauman Ahmed, Jonathan Lévy, Shanshan Ren, Hamid Mushtaq, Koen Bertels, and Zaid Al-Ars. Gasal2: a gpu accelerated sequence alignment library for high-throughput ngs data. *BMC bioinformatics*, 20:1–20, 2019.

[13] Limin Fu, Beifang Niu, Zhengwei Zhu, Sitao Wu, and Weizhong Li. Cd-hit: accelerated for clustering the next-generation sequencing data. *Bioinformatics*, 28(23):3150–3152, 2012.

[14] Hongyi Xin, John Greth, John Emmons, Gennady Pekhimenko, Carl Kingsford, Can Alkan, and Onur Mutlu. Shifted hamming distance: a fast and accurate simd-friendly filter to accelerate alignment verification in read mapping. *Bioinformatics*, 31(10):1553–1560, 2015.

[15] Changyong Yu, Yuhai Zhao, Chu Zhao, Haitao Ma, and Guoren Wang. Diagaf: A more accurate and efficient pre-alignment filter for sequence alignment. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 19(6):3404–3415, 2021.

[16] Taha Shahroodi, Michael Miao, Mahdi Zahedi, Stephan Wong, and Said Hamdioui. Rattlesnakejake: a fast and accurate pre-alignment filter suitable for computation-in-memory. In *International Conference on Embedded Computer Systems*, pages 209–221. Springer, 2023.

[17] Taha Shahroodi, Michael Miao, Joel Lindegger, Stephan Wong, Onur Mutlu, and Said Hamdioui. An in-memory architecture for high-performance long-read pre-alignment filtering. *arXiv preprint arXiv:2310.15634*, 2023.

[18] Mohammed Alser, Hasan Hassan, Hongyi Xin, Oğuz Ergin, Onur Mutlu, and Can Alkan. Gatekeeper: a new hardware architecture for accelerating pre-alignment in dna short read mapping. *Bioinformatics*, 33(21):3355–3363, 2017.

[19] Mohammed Alser, Hasan Hassan, Akash Kumar, Onur Mutlu, and Can Alkan. Shouji: a fast and efficient pre-alignment filter for sequence alignment. *Bioinformatics*, 35(21):4255–4263, 2019.

[20] Zülal Bingöl, Mohammed Alser, Onur Mutlu, Ozcan Ozturk, and Can Alkan. Gatekeeper-gpu: Fast and accurate pre-alignment filtering in short read mapping. In *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 209–209. IEEE, 2021.

[21] Aaron Russell Fajardo, Saira Kaye Manalili, Candace Claire Mercado, Raphael Zapanta, and Roger Luis Uy. Multiprocess implementation of dna pre-alignment filtering using the bit matrix algorithm. In *2020 IEEE 12th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment, and Management (HNICEM)*, pages 1–6. IEEE, 2020.

[22] Mohammed Alser, Taha Shahroodi, Juan Gómez-Luna, Can Alkan, and Onur Mutlu. Sneakysnake: a fast and accurate universal genome pre-alignment filter for cpus, gpus and fpgas. *Bioinformatics*, 36(22-23):5282–5290, 2020.

[23] Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

[24] Heng Li and Nils Homer. A survey of sequence alignment algorithms for next-generation sequencing. *Briefings in bioinformatics*, 11(5):473–483, 2010.

[25] Todd Z DeSantis, Philip Hugenholtz, Neils Larsen, Mark Rojas, Eoin L Brodie, Keith Keller, Thomas Huber, Daniel Dalevi, Ping Hu, and Gary L Andersen. Greengenes, a chimera-checked 16s rrna gene database and workbench compatible with arb. *Applied and environmental microbiology*, 72(7):5069–5072, 2006.

[26] Gürol Canbek, Seref Sagiroglu, Tugba Taskaya Temizel, and Nazife Baykal. Binary classification performance measures/metrics: A comprehensive visualized roadmap to gain new insights. In *2017 International Conference on Computer Science and Engineering (UBMK)*, pages 821–826. IEEE, 2017.

[27] Zhen Ju, Huiling Zhang, Jintao Meng, Jingjing Zhang, Jianping Fan, Yi Pan, Weiguo Liu, Xuelei Li, and Yanjie Wei. ngia: A novel greedy incremental alignment based algorithm for gene sequence clustering. *Future Generation Computer Systems*, 136:221–230, 2022.

VI. Biography Section

Zhen Ju received the B.S. degree in communication engineering from Lanzhou University of Technology, Lanzhou, China, in 2014, and the M.S. degree in computer applications technology from University of Chinese Academy of Sciences, Beijing, China, in 2016. He is currently pursuing the Ph.D. degree at the University of Chinese Academy of Sciences, Beijing, China. His current research interests include heterogeneous computing and high-performance computing.

Jjingjing Zhang received the B.S. degree in biological science from Northeast Agricultural University, Harbin, China, in 2017, and the M.S. degree in bioinformatics from Shaanxi Normal University, Xi'an, China, in 2020. She is currently pursuing the Ph.D.degree at the University of Chinese Academy of Sciences, Beijing, China. Her current research interest is circular RNAs.

Xuelei Li received the B.S. and M.S. degrees in solid geophysics from Jilin University China, Changchun, China, in 2010 and 2013, respectively, and the Ph.D. degree in solid geophysics from Institute of Geodesyand Geophysics, Chinese Academy of Sciences, Wuhan, China, in 2017. He is currently a deputy senior engineer with Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences. His research interests include seismic wave propagation and inversion, and high performance computing.

Jintao Meng received the B.S. and M.S. degrees in computer science from the Central China Normal University, Wuhan, in 2005 and 2008 respectively, and the Ph.D. degree in computer architecture from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, in 2016. He is an research associate with the Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences. He is the author of SWAP-Assembler, and published 20 articles, and 18 inventions. His research interests include high performance computing, bioinformatics, and graph computing.

Yanjie Wei is a professor and the director in Center for High Performance Computing, Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences. He earned his Ph.D. in 2007 at Michigan Tech University and from 2008 to 2011, he worked as a postdoctoral research associate at Princeton University. His research focuses on high performance computing and computational biology/bioinformatics. He has published more than 130 peer reviewed journal/conference papers.