2022 IEEE International Conference on Bioinformatics and Biomedicine (BIBM) | 978-1-6654-6819-0/22/$31.00 ©2022 IEEE | DOI: 10.1109/BIBM55620.2022.9995449

Crescent: A GPU-based Targeted Nanopore Sequence Selector

Tong Li[1], Xueqi Li[2,*], Yewen Li[2,3], Ruibao Song[1], and Xun Wang[1,2,*]
[1]College of Computer Science and Technology, Qingdao Institute of Software, China University of Petroleum, Qingdao, Shandong 266580, China
[2]Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China
[3]University of Chinese Academy of Sciences, Beijing 100049, China
[*]Corresponding Author
Email: ltong@s.upc.edu.cn {lixueqi, liyewen19s}@ict.ac.cn s21070019@s.upc.edu.cn wangsyun@upc.edu.cn

***Abstract*—With the development of three-generation genome sequencing technology, targeted sequencing has become a fundamental need in virus detection, metagenomics expansion, and human polymorphisms detection. However, previous studies on CPU/GPU platforms have much lower throughput than Nanopore sequencers, and hardware designs have high design cost and flexibility issues. In this paper, we propose Crescent, a GPU-based targeted sequence selector. Specifically, on the algorithm level, we modify the popular-used algorithm without significantly decreasing the accuracy and then propose four key observations. On the architecture level, based on four key observations, we propose two-stage parallelism mechanisms (i.e., inter-task and intra-task) and other optimization strategies (i.e., data reuse and redundant computation elimination) to accelerate our modified algorithm. Our experimental results show that the throughput of Crescent on NVIDIA A100 is 10.84× and 2.30× higher than a state-of-the-art 16-thread and 72-thread CPU baseline, respectively.**

***Index Terms*—Oxford Nanopore Sequencing, SDTW, GPU Acceleration**

I. Introduction

Genome sequencing technologies, which determine the DNA sequence of an organism, play an essential role in personalized medicine [1]–[3], [10], evolutionary theory [4], and forensics [5]. Since the contiguous DNA reads produced by Nanopore sequencing can reach approximately hundreds of thousands of bases [6] compared to the hundreds of bases of the traditional second-generation sequencing technology [7]. It has become a promising three-generation sequencing technology, which can significantly improve the quality of sequenced genome assemblies and reveal a broader spectrum of disease and evolutionary mutations [29]–[32].

Since the basecalling process (i.e., converting current signals to base pairs) takes up 30% $\sim$ 90% of the whole genome analysis process, it is essential to sequence only the target genome [18], [38]. Targeted sequencing (aka. Read Until) has become an essential scenario for Nanopore sequencing, which means filtering out unwanted reads and keeping the desired reads [14], [15], [17], [21], [22], [25]. This technique is widely used in virus detection, specific bacteria expansion in the metagenomics community [16], and precise detection of human polymorphisms, structural variants [20], and methylation [33].

Distinct from the outputs of the second-generation sequencing, which are hundred of bases determined by fluorescent labeling [7]. The output of the Nanopore sequencer (e.g., MinION [8]) is a string of current signals characterizing the input bases. Since Nanopore protein pores can hold about six to ten bases, each current value represents more than one base, which poses a challenge for targeted sequencing. Two broad types of studies have been conducted for Nanopore targeted sequencing: (1) basecalling-alignment-based and (2) raw-signal-based. (1) Basecalling-alignment-based refers to the conversion of current signals into bases using basecalling tools (e.g., Bonito [9] and Guppy [11]) followed by alignment software (e.g., BWA-MEM [12] and MiniMap2 [13]) to determine whether a genome is targeted or not. These works [14] are characterized by high accuracy, but the complex software process also limits their latency and scalability. (2) Raw-signal-based has become a hot topic of current research because of its simplicity and efficiency. It directly compares the current signals generated by sequencers with targeted genomes. Loose et al. [15] and SquiggleFilter [18] use sDTW and its variant algorithms to perform direct signal comparison. Sigmap [21] and UNCALLED [17] improve the speed and accuracy of exact matching of specific sub-fragments by building FM-index data structures for targeted genomes in advance. SquiggleNet [19] trains a deep learning model through the datasets of targeted and non-targeted signals in advance and ultimately recognizes targeted sequencing signals. SquiggleFilter [18] proposes a hardware-friendly sDTW algorithm and designs a hardware implementation that provides significant throughput.

However, these methods do not meet the demand for targeted sequencing in terms of throughput and low cost. For example, the throughput of Loose et al. [15] is 40× lower than the throughput of the current MinION sequencer. UNCALLED [17] and Sigmap [21] have improved throughput by introducing the FM-index data structure, but the throughput is still lower than MinION and requires a complex pre-processing process. SquiggleFilter [18] has outstanding improvements in throughput. But the high chip design cost and lack of flexibility

Authorized licensed use limited to: INSTITUTE OF COMPUTING TECHNOLOGY CAS. Downloaded on March 09,2023 at 06:59:11 UTC from IEEE Xplore. Restrictions apply.

become its problems.

To improve throughput and maintain the programmability of this rapidly evolving algorithm, in this paper, we propose Crescent, a GPU-based targeted Nanopore sequence selector. First, we modify the sDTW algorithm of SquiggleFilter [18] and then propose four key observations (noted as Observation-1 to Observation-4). Second, based on Observation-1, we propose Inter-Query Parallelism to exploit the parallelism of the modified sDTW algorithm at the GPU block level, allowing multiple Query to be compared in parallel using a single GPU. Based on Observation-2, we propose Intra-Query Parallelism to leverage the parallelism at the GPU thread level, allowing non-dependency columns to be computed in parallel. Third, we utilize Observation-3 and Observation-4 to further decrease our single iteration latency from the perspective of data reuse and redundant computation elimination.

Our specific contributions are listed as follows.

- We propose a GPU-based targeted nanopore sequence selector called Crescent. We first analyze the rapidly evolving Read Until algorithm and point out the shortcomings of existing algorithms and ASIC implementations. Based on the observation and improvement of the algorithm, our proposed design not only achieves high-throughput acceleration on GPU but also ensures accuracy.
- We propose both inter- and intra-query parallelism designs. For the inter-query design, we propose a query partition method and query-block mapping method to make full use of the limited GPU blocks. For the intra-query design, we propose a specific tiling method and data caching method to improve parallelism as well as memory utilization.
- In addition, we propose query reuse and algorithm-specific redundant computation elimination strategies to reduce memory footprint and parallel reduction, leading to performance improvement.
- Experimental results show that Crescent achieves 172,062 $\sim$ 438,250 bases/s, which is 10.84$\times$ and 2.30$\times$ more than a state-of-the-art CPU baseline with 16 and 72 threads, respectively. The throughput of Crescent is much higher than a Nanopore MinION sequencer (230,400 samples/s) [18], which means that one GPU card can afford the targeted sequencing of two Nanopore sequencers.

II. Background

In this section, we first introduce the essential role of the Read Until process in Nanopore sequencing. Second, we introduce the widely used targeted sequencing algorithm sDTW [18], and please refer to [17], [19], [21], [22] for more details. Finally, we introduce the latest NVIDIA Ampere GPU architecture.

Read Until. Read Until is a feature of the ONT sequencer that analyzes reads (DNA/RNA fragments) in real-time during sequencing. If a non-viral read is detected, the sequencer will eject it, thus saving the time and cost of sequencing non-viral reads ($>$ 99% of all reads). Unfiltered viral reads are used to construct the entire viral genome using reference-guided assembly (alignment and variant calling).

Algorithm 1 sDTW Algorithm [18]

Inputs: R (reference), Q (query)
Outputs: min (minimum matching cost)

```
1:  M ← R.size()
2:  N ← Q.size()
3:  S ← zeros(N, M)
4:  S[0,0] ← abs(Q[0] − R[0])
5:  prev ← zeros(N)
6:  curr ← zeros(N)
7:  bonus ← 10
8:  for i in 1 : N do
9:      S[i,0] ← S[i−1,0] + abs(Q[i] − R[0])
10: for j in 1 : M do
11:     for i in 1 : N do
12:         move ← S[i−1,j−1] − prev[i−1] × bonus
13:                 < S[i−1,j]
14:         if move then
15:             curr[i] ← 0
16:             S[i,j] ← abs(Q[i] − R[j]) + S[i−1,j−1]
17:                     −prev[i−1] × bonus
18:         else
19:             curr[i] ← min(10, prev[i] + 1)
20:             S[i,j] ← abs(Q[i] − R[j]) + S[i−1,j]
21:         prev ← curr
22:         curr ← zeros(N)
23: return min(S[N,:])
```

Algorithm Introduction. The subsequence Dynamic Time Warping (sDTW) algorithm allows classification between the measured signal (Query) and the pre-calculated reference signal (Reference). It allows the entire query signal to be aligned with any small part of the reference signal. The Algorithm 1 shows how to match a signal (Query) to the local position of a reference sequence (Reference). First, lines 1-4 initialize the score matrix S and calculate the first value of the matrix. Second, lines 4-7 introduce two intermediate variables, prev and curr. Third, lines 8-9 initialize the first column of the matrix S. Fourth, lines 10-21 are the process of calculating the minimum matching cost. Finally, return the best result S[N, :]. Since signals with higher average translocation rates generally have higher matching costs. To ensure sDTW matching costs solely to represent the matching quality and are independent of translocation rate, a "match bonus" mechanism is implemented to reward multiple signal matches to the same reference base. A constant (10 in Line 7) is used to represent the upper bound of consecutive matches. As demonstrated in Lines 12-20, the move variable is used to indicate whether the optimal match comes from S[i-1,j-1] or S[i-1,j] (Lines 12-13). If it comes from S[i-1,j], then it means that there are multiple signals aligned to a single reference base (Lines 18-20), and conversely, a single signal aligned to a single reference base (Lines 14-17).

Authorized licensed use limited to: INSTITUTE OF COMPUTING TECHNOLOGY CAS. Downloaded on March 09,2023 at 06:59:11 UTC from IEEE Xplore. Restrictions apply.

NVIDIA Ampere GPU architecture. Graphics processing units (GPUs) were developed for rendering graphics but are now being used to accelerate many other applications due to their massively parallel architecture [35]. Fig. 1(a) shows the classical NVIDIA Ampere architecture. The difference from the classical CPU is that a large amount of SM (e.g., 128 in A100) is used for parallel computation. As shown in Fig. 1(b), the SM is organized in groups into several Sub-Cores. Each SM also includes an L1 cache and shared memory. Fig. 1(c) illustrates the Sub-Core architecture, in which programmers can allocate computing tasks to the computation units (i.e., INT32, FP32, FP64, and Tensor Core), and scheduling will be done jointly by LD/ST, Warp Scheduler, and Dispatch Unit, etc. The functions performed on the GPU are called kernels. The data to be processed by the kernel is first copied from the host memory to the global memory of the GPU. The CPU (called the host) then starts the kernel. Once the kernel is complete, the results are copied from global memory to host memory. Data transfer between CPU and GPU is through PCIe, which is an order of magnitude slower than accessing GPU global memory. Therefore, we should fully use GPU memory to efficiently process large data sets and reduce the data transfer between CPU and GPU [34].

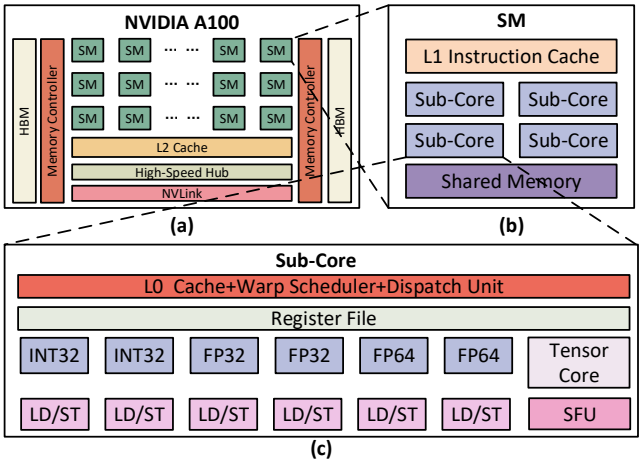

Fig. 1. (a) NVIDIA A100 GPU Architecture; (b) A single Stream Multiprocessor (SM) architecture; (c) A single Sub-Core architecture.

III. Methods

A. Algorithm Modification

To overcome the effect of conversion rate, Algorithm 1 introduces the bonus mechanism of matching consecutive signals to the same reference base. SquiggleFilter [18] shows that the introduction of this mechanism can improve the matching performance of the algorithm by a small sample length, but limits the possibility of accelerating this algorithm on the GPU. As shown in Algorithm 2, we remove the bonus mechanism in order to allow this algorithm to be fully parallelized on the GPU. To compensate for the loss of accuracy due to the removal of the bonus mechanism, we used a longer sample length that can capture more matching information to determine whether it is a targeted genome or not. We evaluate the accuracy of the modified algorithm in Sec. IV and describe the data dependency is described in Sec. III-B.

Algorithm 2 Modified sDTW Algorithm

Inputs: R (reference), Q (query)
Outputs: min (minimum matching cost)

$M \leftarrow R.size()$
$N \leftarrow Q.size()$
$S \leftarrow zeros(N, M)$
$S[0,0] \leftarrow abs(Q[0] - R[0])$
for i in 1 : N **do**
    $S[i,0] \leftarrow S[i-1,0] + abs(Q[i] - R[0])$
for i in 1 : N **do**
    for j in 1 : M **do**
        $S[i,j] \leftarrow abs(Q[i] - R[j]) +$
            $min(S[i-1,j-1], S[i-1,j])$
return $min(S[N,:])$

B. Key Observations

Based on the analysis of Sec. III-A for the modified sDTW algorithm and GPU architecture, we get the following four key observations.

Observation-1. A typical computational scenario of targeted sequencing is that several different queries generated by multiple sequencers are aligned with an identical targeted genome at the same time, and the computation of the different queries is also independent of each other.

Observation-2. From Fig. 2(b), we can see that the result of S[i,j] depends on the previous i iteration to produce S[i-1,j-1] and S[i-1,j], while there is no dependency with S[i,j-1] of the current i iteration.

Observation-3. As shown in Fig. 2(a), the sDTW algorithm contains two iterations, i.e., the outer iteration of Query (noted as i) and the inner iteration of Reference (noted as j). The inner j iteration calculates the distance between each signal of Reference and the current signal of Query. The outer i iteration changes the signal of Query for each computation with Reference. This means that the inner iteration j requires the same Query. In addition, the outer iteration i needs to be executed serially because of the dependency between them.

Observation-4. Since an integer number of samples (i.e., 2000, 5000, and 9000) [18] is usually used for identifying target signals, not all minimum costs need to be calculated.

C. Crescent Design Overview

Based on Observation-1, the computation of the different queries is independent of each other, we use inter-block parallelism of the GPU to perform the matching task of different Query simultaneously (see Sec. III-D). Based on Observation-2, there is no dependency between calculations in the same row, we exploit the intra-block parallelism of the GPU to perform chunked parallelism for a single task (see

Authorized licensed use limited to: INSTITUTE OF COMPUTING TECHNOLOGY CAS. Downloaded on March 09,2023 at 06:59:11 UTC from IEEE Xplore. Restrictions apply.

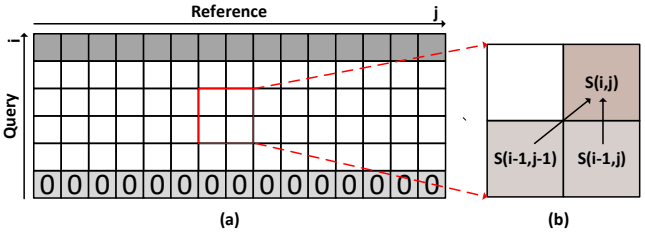

Fig. 2. (a) Algorithm execution for sDTW; (b) Single-iteration dependency (corresponds to the red box in (a)).

Sec. III-E). Based on Observation-3 and Observation-4, the same query fragment needs to be repeated with the reference and the sample length of the final result is discrete, we propose data reuse and redundant computation elimination strategies to further improve the throughput, respectively (see Sec. III-F).

D. Inter-Query Parallelism

A common scenario for target sequencing is to match several different queries from the sequencer with a pre-processed reference (i.e., target genome) and finally output a score for each query to determine whether it is a target genome. As shown in Fig. 3, the Reference sequence and Queries 0-4 are first loaded from global memory into Blocks 0-4, and then the matching process is processed in each block, while all blocks are computed in parallel. In addition, due to the limited number of blocks (e.g., five in our toy example), only part of the sequence can be computed at a time, so we divide all queries into different tasks, Task 1 will be executed after Task 0 is finished in our example. The output of each query determines whether it is a targeted genome, and they are written back to global memory after the computation is finished. Fig. 3 is not shown since the memory cost is small.

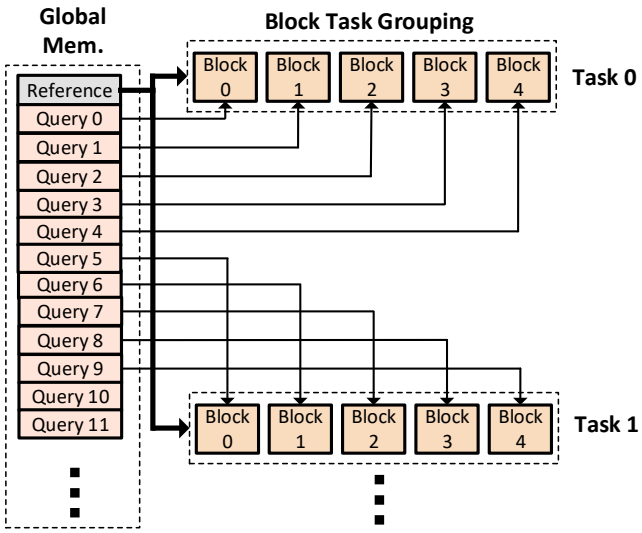

Fig. 3. Inter-query parallelism.

E. Intra-Query Parallelism

Task Tiling. The typical reference genome, such as lambda phage, has a length of 48502 samples [23]. The query is usually only a fragment of the length of the reference, as the fragment is long enough to distinguish whether it is a targeted genome or not. Typically, the query length is taken as 1000 $\sim$ 9000 samples [18]. Due to the limited number of parallel threads within a block (e.g., 1024 in NVIDIA A100) and shared memory (e.g., 48KB in NVIDIA A100), we use the following chunking technique for reference and query.

- Since the number of parallel threads within a block is much smaller than the reference length, the reference sequence needs to be executed into chunks. As shown in Fig. 4(a), the Reference is divided into five fragments, R 0 to R 4. The lengths of R 0 to R 3 are all equal to the number of threads (e.g., 512 or 1024), and the length of R 4 is the final remaining length divided by the number of threads.
- Since the reference in the block is executed literately, subsequent tiles require the results of previous tiles. For example, the computation of the first row of Tile 1 involves the result of the last row of Tile 0. Obviously, longer query iterations will result in more intermediate results to be cached. Due to the limited shared memory within a block (e.g., 48KB in NVIDIA A100), we can not iterate the entire query at once. As shown in Fig. 4(a), the Query is also divided into two separate fragments Q 0 and Q 1.

To summarize, the matching process between a reference sequence and a query sequence is divided into ten sub-tasks serially executed by a block from Tile 0 to Tile 9.

Parallelism in One Tile. Fig. 4(b) gives the execution process of Tile 0 in Fig. 4(a). S(i,j) denotes the computation result of the i-th thread at the j-th iteration. For example, as shown in the first column, T 0 computes S(0,1), T 1 computes S(1,1), T 2 computes S(2,1), and T 3 computes S(3,1), and these four threads compute in parallel to obtain the result in the first column. The result of the first column is stored temporarily. In the second iteration of the threads, the result of the previous iteration is obtained, and then the next column is computed, and so on. Fig. 4(c) depicts the computation from S(i,1) to S(i,2) using four threads. The results of the first iteration are stored in shared memory, and the T 0 - T 3 threads depend on the result of the first column when they compute the second column. The results of four parallel threads are temporarily stored in registers and then written back to shared memory.

Since each thread has strictly the same computation overhead and has no load imbalance problem, we do not introduce a finer-grained warp scheduling mechanism. We specify the number of threads directly by the number of rows in the tile, and take the number of threads to be an integer multiple of 32 to ensure efficient warp scheduling.

Data Caching for Tiles. To support the tiling technique shown in Fig. 4(a), the computation in each Tile needs to

Authorized licensed use limited to: INSTITUTE OF COMPUTING TECHNOLOGY CAS. Downloaded on March 09,2023 at 06:59:11 UTC from IEEE Xplore. Restrictions apply.

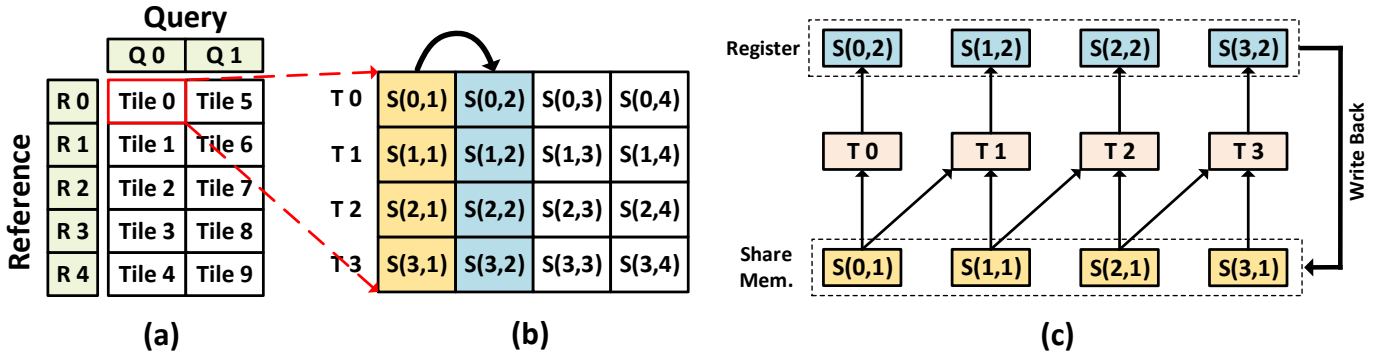

Fig. 4. Intra-query parallelism. (a) Block iteration for a single query; (b) Thread-level parallelism of Tile 0 blocks; (c) Computation and data movement of the first iteration.

cache some intermediate results. We describe how to cache these data reasonably in the following.

- Since the computation of the first line of Tile 1 depends on the last line of Tile 0 (the first lines of other Tiles has the same phenomenon), we need to cache S(3,j) as shown in Fig. 4(b) into shared memory at the end of each iteration. Since we chunk the query in advance, the overhead is acceptable (e.g., about 4KB for a query chunk length of 1000).
- Since the computation of the first column of Tile 5 depends on the last column of Tile 0 (the first columns of the other Tiles has the same phenomenon), we need to cache the S(i,4) in Fig. 4(b). The computation of Tile 5 to Tile 9 needs to be started after the completion of Tile 0 to Tile 4, and the reference sequence length is usually greater than 20000. Therefore, writing these data back to shared memory is not acceptable. We choose to write these data to global memory. Since these data are only written back to global memory in the last column of the tile and read in the first column, the long access latency to the global memory will not affect the performance too much.
- As shown in Fig. 4(a), the computation of the top-leftmost element of Tile 6 depends not only on Tile 1 and Tile 5 but also on the bottom-rightmost corner element of Tile 0 [1]. Considering that the dependent element has only one value, which is already loaded when Tile 5 is computed, we can cache this particular value in a private variable after the computation of Tile 5 is finished.

F. Other Optimization Strategies

Query Reuse. As shown in Fig. 4(a), the order of tile computation is 0 to 9. This means that the reference sequence is executed in fragments, and the same query data needs to be computed against different fragments of the reference. This inspires us to reduce the data loading time and improve bandwidth utilization by reusing query sequences. As shown in Fig. 5, we reuse the Q 0 in the computation of Tile 0 to Tile 4 and go by iteration to load R 0 to R 4 of the Reference. Similarly, we reuse the Q 1 in the computation for Tile 5 to Tile 9.

[1]This phenomenon also appears on several blocks from Tile 7 to Tile 9.

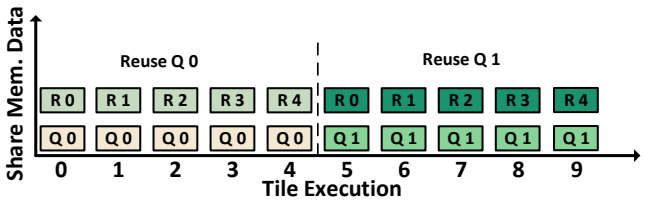

Fig. 5. Query reuse in intra-signal parallelism.

Redundant Computation Eliminate. The purpose of the sDTW algorithm is to find the best matching for the query and reference sequences, so we need to find the minimum value of each column after computation. Using parallel thread reduction in GPU to find the minimum value of each column is very time-consuming, which will seriously affect the efficiency of column iteration. We found that since we only need to get the minimum value of a specific column (e.g., 1000, 5000, and 9000) [18], we do not need to compute the minimum value of all columns. As shown in Fig. 6, since we only need to get the minimum value of column S(i,4), all three columns from S(i,1) to S(i,3) can eliminate the finding minimum process. We use thread reduce to perform the parallel computation to find the minimum of the column S(i,4).

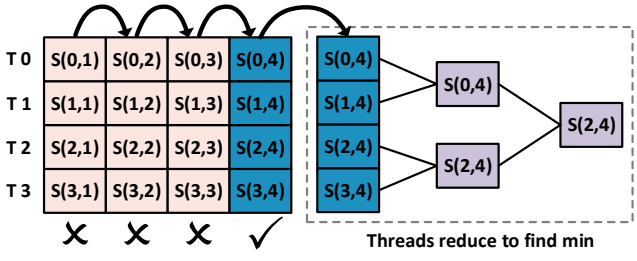

Fig. 6. Eliminate redundant finding minimum computation.

Authorized licensed use limited to: INSTITUTE OF COMPUTING TECHNOLOGY CAS. Downloaded on March 09,2023 at 06:59:11 UTC from IEEE Xplore. Restrictions apply.

IV. Evaluation Results

A. Methodology

Benchmark Datasets. We used five datasets to evaluate the performance of Crescent, which are Lambda Phage (DB1), Human mastadenovirus B (DB2), Human coronavirus OC43 (DB3), SARS-CoV-2 (DB4), and Tai Forest ebolavirus (DB5). The DB1 and DB4 datasets contain MinION R9.4.1 data from SquiggleFilter [18] and the Cadde Centre [23]. We obtained DB2, DB3, and DB5 reference sequence data from NCBI [28] and then used DeepSimulator 1.5 [27] to simulate the MinION R9.4.1 current data. We also used Human DNA datasets [24] from the ONT open dataset to evaluate the accuracy of Crescent.

Baseline Platform. Our first evaluation platform is an RHEL 7 with a 36-cores 72-threads Intel(R) Xeon(R) Gold 6354 CPU @3.00GHz with 256 GB host memory and an NVIDIA A100 GPU with 32GB HBM memory, which are noted as Crescent-CPU-72 and Crescent-GPU-A100 in the following paper. Specifically, the NVIDIA A100 GPU has 6912 CUDA Cores and 432 Tensor Cores, and a boost frequency of 1.41GHz. Our second evaluation platform is a Ubuntu server (18.04) with an 8-cores 16-threads Intel(R) Core(TM) i7-10700F @2.90GHz with 110 GB host memory and an NVIDIA Ampere RTX3090 GPU with 24GB GDDR5 memory, which are noted as Crescent-CPU-16 and Crescent-GPU-3090. Specifically, the NVIDIA Ampere RTX3090 GPU has 10496 CUDA Cores and a boost frequency of 1.70GHz.

Crescent is implemented with C++ and CUDA C. The GPU device kernel is compiled with CUDA (v11.3), and the CPU host code is compiled with GCC 7.5.0 with the compilation option of "-std=c++14 -O3". We evaluate the open source code of SquiggleFilter [18] on two platforms, and they are denoted as SquiggleFilter-CPU-16 and SquiggleFilter-CPU-72, respectively.

B. Overall Results

Crescent Throughput on CPU and GPU. Fig. 7(a) shows the throughput of the CPU and GPU implementation of Crescent. Since Crescent-CPU-16 and Crescent-CPU-72 only leverage the parallelism between queries, they both suffer from sDTW being a compute-bound task, and thus both have relatively low performance. The performance of Crescent-CPU-16 on DB1 to DB5 is only 2333 $\sim$ 5000 samples/s due to the low parallelism. Compared to Crescent-CPU-16, the performance of Crescent-CPU-72 on five data sets is about 4$\times$ $\sim$ 5$\times$ of Crescent-CPU-16. This is due to Crescent-CPU-72 having a quadruple number of threads and a relatively high processor frequency compared to Crescent-CPU-16. Since Crescent exploits parallelism between and within queries and fully leverages data reuse and computation elimination, it achieves significant performance gains over CPUs. We execute 10,000 queries for five datasets on two GPUs. The experimental results showed that the Crescent-GPU-3090 performed 130507 $\sim$ 333873 samples/s on DB1 to DB5 and the Crescent-GPU-A100 performed 172062 $\sim$ 438250 samples/s on DB1 to DB5, achieving about 13$\times$ and 18$\times$ throughput improvements respectively compared to the Crescent-CPU-72. Although the performance of a Sub-Core is lower than a single CPU core, Crescent improves performance by increasing inter-query parallelism (e.g., 128 in NVIDIA A100) and intra-query parallelism (e.g., 1024 in NVIDIA A100). Considering running Crescent on different GPUs, the Crescent-GPU-A100 achieves a performance increase of 1.31$\times$ compared to the Crescent-GPU-3090 has a higher memory bandwidth.

Crescent Throughput vs SquiggleFilter. Fig. 7(b) compares Crescent with the CPU implementation of SquiggleFilter [18]. Since the software of SquiggleFilter uses Python multiprocessing and a high-performance numba implementation, it outperforms about 8$\times$ compared to the CPU of Crescent. The throughput of Crescent-GPU-3090 and Crescent-GPU-A100 are 8.26$\times$ and 10.84$\times$ higher than the throughput of SquiggleFilter-CPU-16, respectively, and 1.75$\times$ and 2.3$\times$ higher than SquiggleFilter-CPU-72. This is because Crescent exploits both block-level and threads-level parallelism on the GPU (about 128 $\times$ 1024), effectively counteracting the low performance of a single Sub-Core. In addition, data reuse and redundant computation elimination also reduce the impact of low single-core performance.

Accuracy. Crescent eliminates the bonus mechanism compared to the original sDTW algorithm [18], and we evaluate whether the modified algorithm still has better classification performance by measuring the F-score. Fig. 8 shows the maximum F-score of our algorithm on five different datasets, DB1-DB5. As the sample length increases, Crescent can distinguish DB1-DB5 and human DNA fragments accurately, and the classification accuracy is gradually increasing when it has a longer sample length. The commonly used sample length 5000 has 90% in accuracy on both DB1-DB5. It can be seen that although Crescent modified the algorithm, the classification accuracy is still high.

C. Optimization Analysis

Impact of the Number of Threads. The degree of GPU parallelism depends on the block size, which means the number of threads that are paralleled in one block. The maximum number of parallel threads per block in the NVIDIA Ampere GPU architecture is 1024. As shown in Fig. 9, Crescent achieves maximum throughput on the five datasets DB1-DB5 when the block size is 1024. And as the block size gradually gets smaller, the number of parallel threads decreases until the block size is 128, and there is no significant change in throughput. This is because although the number of threads per block decreases, the GPU can start more blocks, achieving almost the same parallelism. Moreover, SM execution is performed in warps (e.g., 32 in A100), and a too large block size does not improve performance and is limited by the warp scheduling efficiency. As shown in Fig. 9, a significant reduction in throughput occurs as the number of threads per block drops to 64 and below. Although a smaller thread number allows a single SM to provide more blocks, the shared memory of each SM has an upper bound, and a

Authorized licensed use limited to: INSTITUTE OF COMPUTING TECHNOLOGY CAS. Downloaded on March 09,2023 at 06:59:11 UTC from IEEE Xplore. Restrictions apply.

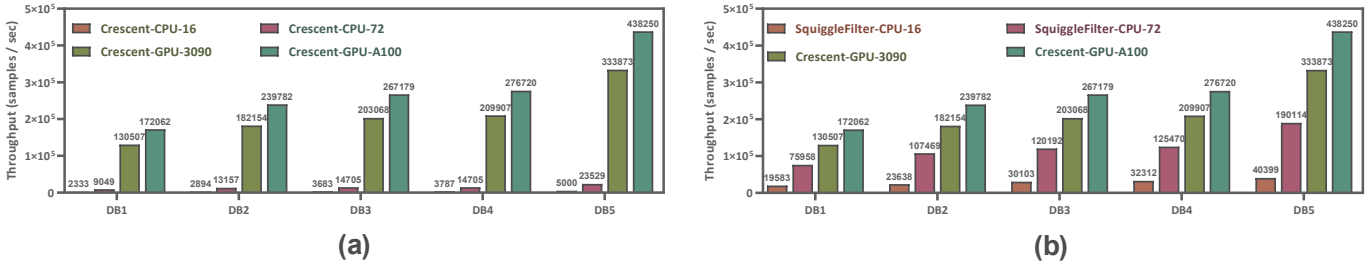

Fig. 7. The throughput comparisons on different platforms and datasets. (a) Crescent CPU vs. Crescent GPU; (b) SquiggleFilter [18] CPU vs. Crescent GPU.

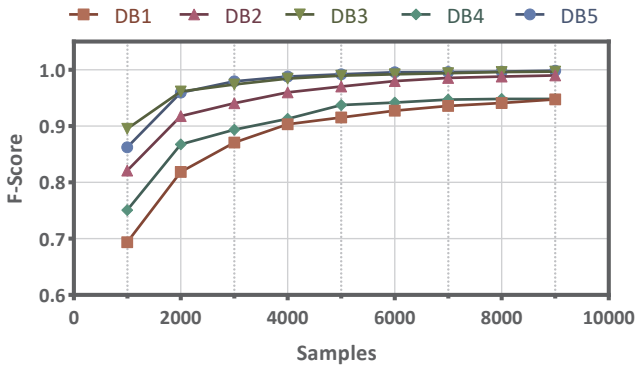

Fig. 8. Classification accuracy of Crescent on different datasets and sample lengths.

portion of shared memory per block is consumed independent of the block size (about 9KB when the query fragment length is 1000), this results in a single SM not being able to utilize all threads in parallel, thus affecting performance.

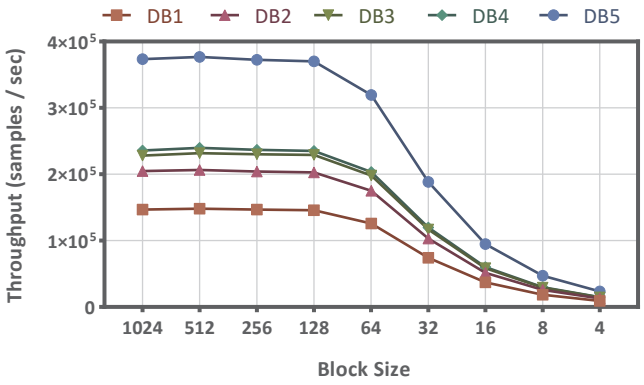

Fig. 9. The impact of different numbers of threads on the throughput of Crescent.

Impact of Redundant Computation Elimination. Crescent uses the redundant computation elimination when computing the matching cost, specifying the interval at which the minimum value of a column needs to be computed. As shown in Fig. 10, the redundancy elimination strategy is not used when the value of the sample scale is 1, i.e., the minimum value of each column is computed. The throughput of the five datasets DB1-DB5 is at the lowest level because computing the minimum is time-consuming and dominates the column iteration time. As the sample scale is increased from 1 to 2000, it can be seen that the throughput will gradually increase with the value of the sample scale. In practice, the sample scale is 1000 or 2000, which will allow the redundancy elimination mechanism to improve the performance by about 4× and have no loss of accuracy.

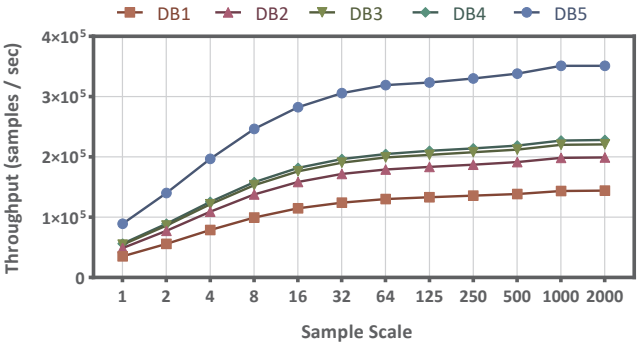

Fig. 10. The impact of redundant computation elimination strategies on the throughput of Crescent.

V. Related Work

After Nanopore published its first sequencer in 2014, Loose et al. [15] proposed the first software for targeted sequencing. This work uses DTW and sDTW for the targeted sequencing process for the first time, but the throughput using an 8-core ARM processor is 40× slower than the MinION sequencer. With the accuracy and speed of the basecalling algorithm improving, some works [14], [26], [36], [37] use basecalling followed by alignment for targeted sequencing. This type of method can achieve high accuracy but latency and scalability become bottlenecks. Inspired by the alignment software (e.g., BWA-MEM [12] and MiniMap2 [13]), Sigmap [21] and UNCALLED [17] split the targeted sequencing into three processes: event segmentation, FM-index search, and seed clustering. Although these methods are more accurate and efficient, they require the generation of pre-processing files (about 3.2 GB) and a long sample length [18]. SquiggleFilter [18] first used the domain specific architecture to accelerate the process, achieving astonishing speedup ratios and respectable accuracy rates, but the hardware design has an expensive cost and is poorly programmable. In this paper, we use GPU for the first

Authorized licensed use limited to: INSTITUTE OF COMPUTING TECHNOLOGY CAS. Downloaded on March 09,2023 at 06:59:11 UTC from IEEE Xplore. Restrictions apply.

time to accelerate the targeted sequencing process based on Loose et al. [15] and SquiggleFilter [18]. On a single server-class GPU, Crescent achieved satisfactory results in terms of both accuracy and throughput, which can simultaneously support the targeted sequencing of two MinION sequencers [18].

VI. Conclusion and Future Work

The rapid development of Nanopore genome sequencing technology has made the acceleration of genome analysis processes a hot research topic in computer architecture and biology. In this paper, we propose Crescent accelerate the targeted sequencing process. Our design consists of two parts. First, we modify the traditional algorithm to give the possibility of acceleration on GPU. Then, we propose four strategies (i.e., inter-query parallelism, intra-query parallelism, data reuse, and redundant computation elimination) to accelerate the modified algorithm. The experimental results demonstrate that the throughput of Crescent outperforms the state-of-the-art 16-thread CPU and 72-thread CPU baseline at about 10.84× and 2.30× on the NVIDIA A100, respectively. We plan to implement the new algorithm on some low-end GPUs (i.e., NVIDIA Jetson AGX Orin) to demonstrate that average labs with limited computing resources can benefit from this advancement. We also plan to develop a CPU-GPU hybrid method to further improve the throughput of Crescent in the future.

VII. Acknowledgements

We thank the anonymous reviewers for their valuable feedback and insightful comments. This work was supported by the NSF of China (No.61873280, No.61972416, 62202454), the Shandong Provincial Natural Science Foundation (No.ZR2019MF012), the Taishan Scholar Foundation (No.ZX20190157), and the China National Postdoctoral Program for Innovative Talents (No. BX2021320).

References

[1] C. Alkan, J. M. Kidd, T. Marques-Bonet, G. Aksay, F. Antonacci, F. Hormozdiari, J. O. Kitzman, C. Baker, M. Malig, O. Mutlu, S. C. Sahinalp, R. A. Gibbs, and E. E. Eichler, "Personalized Copy Number and Segmental Duplication Maps Using Next-Generation Sequencing," Nature Genetics, 2009.

[2] S. Pang, Y. Zhang, T. Song, X. Zhang, X. Wang, and R. Alfonso, "AMDE: a novel attention-mechanism-based multidimensional feature encoder for drug–drug interaction prediction," Briefings in Bioinformatics, 2021.

[3] T. Song, Y. Zhong, M. Ding, R. Zhao, and Y. Deng, "Repositioning Molecules of Chinese Medicine to Targets of SARS-Cov-2 by Deep Learning Method," 2020 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), pp. 2306-2312, 2020.

[4] H. Ellegren, "Genome Sequencing and Population Genomics in Non-Model Organisms," Trends in Ecology & Evolution, 2014.

[5] M. J. Alvarez-Cubero, M. Saiz, B. Martínez-García, S. M. Sayalero, C. Entrala, J. A. Lorente, and L. J. Martinez-Gonzalez, "Next Generation Sequencing: An Application in Forensic Sciences?," Annals of Human Biology, 2017.

[6] M. Jain, S. Koren, K. H. Miga, J. Quick, A. C. Rand, T. A. Sasani, J. R. Tyson, A. D. Beggs, A. T. Dilthey, I. T. Fiddes, S. Malla, H. Marriott, T. Nieto, J. O'Grady, H. E. Olsen, B. S. Pedersen, A. Rhie, H. Richardson, A. R. Quinlan, T. P. Snutch, L. Tee, B. Paten, A. M. Phillippy, J. T. Simpson, N. J. Loman, and M. Loose, "Nanopore sequencing and assembly of a human genome with ultra-long reads," Nat Biotechnol, vol. 36, no. 4, pp. 338–345, Apr. 2018.

[7] J. Shendure and H. Ji, "Next-generation DNA sequencing," Nat Biotechnol, vol. 26, no. 10, pp. 1135–1145, Oct. 2008.

[8] M. Jain, H. E. Olsen, B. Paten, and M. Akeson, "The Oxford Nanopore MinION: delivery of nanopore sequencing to the genomics community," Genome Biol, vol. 17, no. 1, p. 239, Nov. 2016.

[9] "Bonito." Oxford Nanopore Technologies, Feb. 24, 2022. Accessed: Mar. 02, 2022. [Online]. Available: https://github.com/nanoporetech/bonito.

[10] S. Wang, Z. Du, M. Ding, R. Zhao, A. Rodriguez-Paton and T. Song, "LDCNN-DTI: A Novel Light Deep Convolutional Neural Network for Drug-Target Interaction Predictions," 2020 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), pp. 1132-1136, 2020.

[11] R. R. Wick, L. M. Judd, and K. E. Holt, "Performance of neural network basecalling tools for Oxford Nanopore sequencing," Genome Biol, vol. 20, no. 1, p. 129, Jun. 2019.

[12] H. Li, "Aligning Sequence Reads, Clone Sequences and Assembly Contigs with BWA-MEM," arXiv:1303.3997 [q-bio.GN], 2013.

[13] H. Li, "Minimap2: Pairwise Alignment for Nucleotide Sequences," Bioinformatics, 2018.

[14] H. S. Edwards, R. Krishnakumar, A. Sinha, S. W. Bird, K. D. Patel, and M. S. Bartsch, "Real-Time Selective Sequencing with RUBRIC: Read Until with Basecall and Reference-Informed Criteria," Sci Rep, vol. 9, no. 1, p. 11475, Dec. 2019.

[15] M. Loose, S. Malla, and M. Stout, "Real-time selective sequencing using nanopore technology," Nat Methods, vol. 13, no. 9, pp. 751–754, Sep. 2016.

[16] T. Song, X. Wang, X. Li, and P. Zheng, P, "A programming triangular DNA origami for doxorubicin loading and delivering to target ovarian cancer cells," Oncotarget, vol. 5, 2017.

[17] S. Kovaka, Y. Fan, B. Ni, W. Timp, and M. Schatz, "Targeted nanopore sequencing by real-time mapping of raw electrical signal with UNCALLED," Nature biotechnology, 2020.

[18] T. Dunn, H. Sadasivan, J. Wadden, K. Goliya, K. Y. Chen, D. Blaauw, R. Das, and S. Narayanasamy, "SquiggleFilter: An Accelerator for Portable Virus Detection," in MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture, New York, NY, USA, pp. 535–549, Oct. 2021.

[19] Y. Bao, J. Wadden, J. R. Erb-Downward, P. Ranjan, W. Zhou, T. L. McDonald, R. E. Mills, A. P. Boyle, R. P. Dickson, D. Blaauw, and J. D. Welch, "SquiggleNet: real-time, direct classification of nanopore signals," Genome Biol, vol. 22, no. 1, p. 298, 2021.

[20] X. Wang, D. Liu, J. Zhu, R. Alfonso, and T. Song, "CSConv2d: A 2-D Structural Convolution Neural Network with a Channel and Spatial Attention Mechanism for Protein-Ligand Binding Affinity Prediction," Biomolecules, vol. 11, no. 5, Apr. 2021.

[21] H. Zhang, H. Li, C. Jain, H. Cheng, K. F. Au, H. Li, and S. Aluru, "Real-time mapping of nanopore raw signals," Bioinformatics, vol. 37, pp. i477–i483, 2021.

[22] A. Payne, N. Holmes, T. Clarke, R. Munro, B. J. Debebe, and M. Loose, "Readfish enables targeted nanopore sequencing of gigabase-sized genomes," Nat Biotechnol, vol. 39, no. 4, pp. 442–450, Apr. 2021.

[23] National Center for Biotechnology Information (NCBI) Datasets, "Lambdavirus lambda, a species of dsDNA virus in the class Caudoviricetes," https://www.ncbi.nlm.nih.gov/data-hub/taxonomy/10710/.

[24] "ONT Open Datasets: Publicly Available Reference Samples GM24385 Dataset Release," https://registry.opendata.aws/ont-open-data/, 2020.

[25] X.Li, T. Song, Z. Chen, X. Shi, C. Chen, and Z. Zhang, "A universal fast colorimetric method for DNA signal detection with DNA strand displacement and gold nanoparticles," Journal of Nanomaterials, 2015.

[26] "Oxford Nanopore Technologies: Read Until API (Application Programming Interface)," https://github.com/nanoporetech/read_until_api/, 2021.

[27] Y. Li, S. Wang, C. Bi, Z. Qiu, M. Li, and X. Gao, "DeepSimulator1.5: a more powerful, quicker and lighter simulator for Nanopore sequencing," Bioinformatics, vol. 36, no. 8, pp. 2578–2580, Apr. 2020.

[28] National Center for Biotechnology Information (NCBI), "Genome." https://www.ncbi.nlm.nih.gov/data-hub/genome/ (accessed Aug. 16, 2022).

[29] O. Spichenok, Z. M. Budimlija, A. A. Mitchell, A. Jenny, L. Kovacevic, D. Marjanovic, T. Caragine, M. Prinz, and E. Wurmbach, "Prediction of eye and skin color in diverse populations using seven SNPs," Forensic Science International: Genetics, vol. 5, no. 5, pp. 472–478, 2011.

[30] X. Li, Z. Wei, B. Wang, and T. Song, "Stable DNA Sequence Over Close-Ending and Pairing Sequences Constraint," Frontiers in Genetics, vol. 12, 2021.

Authorized licensed use limited to: INSTITUTE OF COMPUTING TECHNOLOGY CAS. Downloaded on March 09,2023 at 06:59:11 UTC from IEEE Xplore. Restrictions apply.

[31] R. H. Waterson, E. S. Lander, R. K. Wilson, et al, "Initial sequence of the chimpanzee genome and comparison with the human genome," Nature, vol.437, pp. 69-87, 2005.
[32] C. Y. McLean, P. L. Reno, A. A. Pollen, A. I. Bassan, T. D. Capellini, C. Guenther, V. B. Indjeian, X. Lim, D. B. Menke, B. T. Schaar, A. M. Wenger, G. Bejerano, and D. M. Kingsley, "Human-specific loss of regulatory dna and the evolution of human specific traits," Nature, vol. 471, pp. 216–219, 2011.
[33] S. Wang, D. Liu, M. Ding, Z. Du, Y. Zhong, T. Song, J. Zhu, and R. Zhao, "SE-OnionNet: a convolution neural network for protein–ligand binding affinity prediction," Frontiers in Genetics, vol. 11, 2021.
[34] Z. Wen, J. Shi, B. He, J. Chen, K. Ramamohanarao and Q. Li, "Exploiting GPUs for Efficient Gradient Boosting Decision Tree Training," IEEE Transactions on Parallel and Distributed Systems, vol. 30, no. 12, pp. 2706-2717, Dec. 2019.
[35] S. Shekofteh, H. Noori, M. Naghibzadeh, H. Fröning and H. S. Yazdi, "cCUDA: Effective Co-Scheduling of Concurrent Kernels on GPUs," IEEE Transactions on Parallel and Distributed Systems, vol. 31, no. 4, pp. 766-778, April. 2020.
[36] P. Alexander, H. Nadine, C. Thomas, M. Rory, D. Bisrat and L. Matthew, "Nanopore adaptive sequencing for mixed samples, whole exome capture and targeted panels," BioRxiv, 2020.
[37] R. Ronan, "Oxford Nanopore Technologies (ONT): "Read Until" adaptive sampling," https://nanoporetech.com/resource-centre/read-until-adaptive-sampling, 2022.
[38] D. Senol Cali, J. S. Kim, S. Ghose, C. Alkan, and O. Mutlu, "Nanopore sequencing technology and tools for genome assembly: computational analysis of the current state, bottlenecks and future directions," Briefings in Bioinformatics, vol. 20, no. 4, pp. 1542–1559, Jul. 2019.

Authorized licensed use limited to: INSTITUTE OF COMPUTING TECHNOLOGY CAS. Downloaded on March 09,2023 at 06:59:11 UTC from IEEE Xplore. Restrictions apply.