Accelerating Dynamic Time Warping Algorithm for Nanopore Selective Sequencing Using GPUs

Denuke Dissanayake
Department of Computer Engineering
University of Peradeniya
Peradeniya, Sri Lanka
e16089@eng.pdn.ac.lk

Nipun Dewanarayane
Department of Computer Engineering
University of Peradeniya
Peradeniya, Sri Lanka
e16360@eng.pdn.ac.lk

Maneesha Randeniya
Department of Computer Engineering
University of Peradeniya
Peradeniya, Sri Lanka
e16313@eng.pdn.ac.lk

Hasindu Gamaarachchi
Garvan Institute of Medical Research
Sydney, Australia
hasindu@garvan.org.au

Roshan Ragel
Department of Computer Engineering
University of Peradeniya
Peradeniya, Sri Lanka
roshanr@eng.pdn.ac.lk

***Abstract*—Dynamic time warping is a popular algorithm for determining the ideal alignment of two-time series. Issues with speech recognition were first developed in the 1960s and became popular in the 1970s. Any data turned into a linear sequence, including graphics, audio, and video can be subject to DTW. The DTW algorithm has a time complexity of O(n^2). The DTW algorithm's potential is constrained by its complexity. DTW requires additional time and computer resources when the time series is extremely long. The time series research community appears to have concluded during the past ten years that DTW can accelerate and accomplish correct implementations with less time and space complexity. The algorithm can be improved and expedited because of its simplicity and dynamic programming foundation. There are further benefits when using certain DTW versions specifically tailored for particular applications. DTW has the potential to become stronger and more effective, but the community still does not completely recognize it.**

***Index Terms*—Accelerate Dynamic Time Warping, Nanopore Sequencing**

I. Introduction

A set of instructions called an algorithm is used to carry out a particular task or resolve an issue. They are primarily employed in computer science and mathematics. Algorithms are used in computer science to execute a certain task or software. The algorithm should be appropriate when selecting an algorithm to carry out a task or solve a problem. When selecting an algorithm, there are some qualities to take into account. Complexity, precision, and effectiveness in time and space are some of them.

One of the most crucial algorithms for evaluating the alignments of two-time series is the Dynamic Time Warping algorithm (DTW). The DTW algorithm's ability to measure two-time series with various lengths and time shifts is one of its benefits. It has numerous applications, including shape matching, pattern recognition, signature recognition, and speech recognition. Numerous software and hardware implementations have been developed to optimise the DTW algorithm. In this review, we will go into further detail. The DTW method is utilized in numerous applications in addition to that.

Nanopore Selective Sequencing is one application that uses the DTW algorithm and allows choosing a specific sequence region from a very vast sequence (RNA or DNA). Nanopore Technologies unveiled the MinION sequencer in 2014. In light of that, nanopore-selective sequencing is discussed. Due to its distinctive characteristics, the Nanopore Selective Sequencing technology has become more well-known worldwide. We will discuss this more in the latter part of the paper.

II. Related works

A. Dynamic Time Warping Algorithm(DTW Algorithm)

The mathematical procedure Dynamic Time Warping(DTW) [1] [2] was developed in the 1960s and examined the alignments and similarities between two-time series. The computational difficulty is O(n^2). Because sequences could be analyzed despite shifts and time-series distortions, the technique proved to be useful. It was employed for voice recognition and pattern identification in its early phases. Later on, however, it was used for various other things, including letters, signatures, gesture recognition, handwriting analysis, computer surveillance, data mining pattern recognition, sequence identification, and more.

The distance matrix C $\in$ RN×M reflecting all pairwise distances between two sequences is first constructed by the DTW technique. The local cost matrix for aligning two sequences is known as this distance matrix. After constructing the local cost matrix, the method identifies the alignment path that passes through the low-cost regions of the cost matrix. This DTW-built route is made up of a series of points. The cost matrix is an n-by-n matrix. Each cell's value is determined using the results of the preceding three computations. You can find the warping path by going back over the calculated numbers. The algorithm's time and spatial complexity is O(n^2).

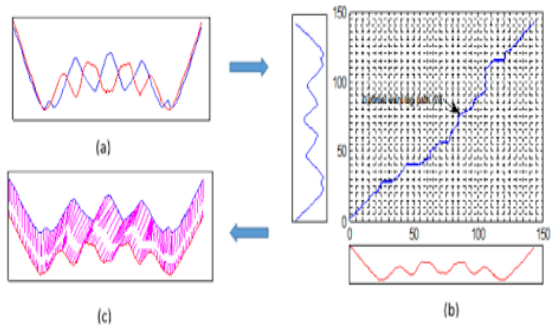


Fig. 1: How the DTW algorithm works. (a) Two-time series from the Plane time-series data set, (b) The optimal warping path and (c) The alignment between these two-time series [3]

B. Applications of DTW Algorithm

This section discusses the DTW algorithm's uses in many fields. We will concentrate more on Nanopore Selective Sequencing and how DTW is used with it from those applications.

1) Speech and Voice Command Recognition: In the 1970s, the Dynamic Time Warping algorithm was frequently employed in voice recognition. By transforming the sound into an electronic signal that an audio device can record, speech recognition is the process of identifying the words that people speak. Then convert them into data that computers can understand. To identify each word separately, the voice stream is converted into integers. The DTW algorithm is used to identify commands and voices. A sample collection of unidentified signals will be compared to template signals by DTW. Each pattern (voice signal) in this example has a unique waveform. The input waveform is compared to each pattern to determine the similarities between the patterns and the existing voice signals [4] [5].

2) Data mining: Clustering and classification are key components of data mining. The accuracy of the distance measuring method directly affects the performance of the clustering and classification algorithms. The overall accuracy can be increased by using a precise distance-measuring algorithm. Recently, the research community has become increasingly interested in DTW as the top algorithm for estimating distance. According to the community, DTW still has room for improvement in performance and accuracy, and certain optimisations are wonderful additions to the traditional DTW [6] [7] [8].

3) Signature verification: Signatures are unique types of handwriting that include intricate geometric designs and words and letters that may not be easily recognized. If two signatures have more differences than matches, it will be determined that one was copied or written incorrectly regarding signature verification [9].

4) Word recognition: Since its release, the DTW algorithm has gained popularity in speech recognition applications. Subsequently, word recognition applications for the DTW method were found to be feasible. Isolated, linked, and discrete word recognition are numerous uses for word recognition [10].

5) Nanopore Sequencing: By reading individual nucleotides when a DNA strand travels through a nanopore, nanopore sequencing is a sort of DNA sequencing technique. Oxford Nanopore Technologies created the technique, which has found widespread use in DNA sequencing applications. A single-stranded DNA molecule is transported through a tiny pore built into a thin membrane during nanopore sequencing. When a pore is subjected to an electric current, it fluctuates as different nucleotides travel through it. It is possible to read the DNA sequence by identifying each nucleotide using the changes in current [11] [12].

C. Nanopore Sequencing

Nanopore sequencing has a long history that dates back to the 1980s. Nanopore Sequencing has undergone numerous advancements since that time to the present. The idea comprises a driving motor protein, DNA or RNA molecules, and a nanopore (a nanoscale protein pore in size). The nanopore encased in a polymer membrane receives a steady voltage. The motor protein is then used to push negatively charged DNA or RNA molecules from the negative side ('cis') to the positive side ('trans') of the nanopore. The motor protein can be used to control speed. The molecule is gradually pushed through the nanopore by the motor protein. The current corresponding to the nucleic acid in the DNA or RNA molecule changes when molecules go through the nanopore. These alterations in the nucleonic sequence's passing region can be tracked and deciphered using mathematical methods [11] [12].

There have been numerous developments in nanopore sequencing over time. One of the innovations was the release of Oxford Nanopore Technologies' MinION sequencer in 2014. (ONT). ONT's MinION has greater benefits over Pacific Biosciences (PacBio) MinION, including portability and a smaller, more compact form. Before to then, PacBio's technology was used in sequencing centres and equipment that were significantly larger. In comparison, MinION is a tiny, four-inch-long USB-powered device. People could purchase a MinION and perform their sequencing because it was less expensive than maintaining servers. The MinION sequencer could be easily connected to a laptop, and the output data could be quickly stored on an external hard drive for later use. The MinION toolbox could be carried to the lab or the fields, whereas earlier samples had to be brought to the sequencing centres to conduct the research. Compared to PacBio technology, this is a significant benefit. The fact that MinION provides real-time information is another important advantage [13] [14].

ONT has a unique function called "nanopore selective sequencing" that enables users to choose a particular sequence from a DNA or RNA molecule. The ONT introduced the "Read Until" Application Programming Interface (API) to achieve selective sequencing. It allows the user to choose only a certain molecule region by switching the voltage applied across the nanopore in the opposite direction and rejecting a certain sequence as needed. Real-time data-gathering capabil-

ities are made possible via selective sequencing. As a result, there is no requirement to wait until the entire DNA or RNA molecule has been sequenced [13] [15].

D. Nanopore Sequence analysing methods

Nanopore sequencing does not require PCR amplification and can provide exceptionally long reads, often between 12 and 120 kbp. There are k consecutive nucleotides in a pore at each time point (denoted as a k-mer, where k is often 5 or 6). Each pore time point's electrical current signal is monitored. Raw electrical current will be produced throughout this operation. There are primarily two techniques to compare a DNA sequence to a reference sequence.

Base calling is the most common method. This base calling method connects the relevant nucleotide sequence to an electrical signal value. The signal will then produce a nucleotide sequence after that. ONT has created Many base callers for nanopore sequence data, first using hidden Markov models and accessible via the Metrichor cloud service. They were swapped out for neural network models that ran on CPUs and GPUs. ONT offers a variety of computational platforms with integrated GPUs for real-time base-calling (minIT, Mk1C, GridION and PromethION). These gadgets make real-time base-calling possible so that flow cells can continue to generate data. These base callers obtained the server-client configuration most recently, allowing the raw signal to be sent to the server and a nucleotide sequence to be returned [16] [17].

End-to-end mapping between the raw electrical signal sequence and the reference sequence signal is another technique for nanopore sequence analysis. Dynamic Time Warping is the most popular and effective method for determining the best mapping for signal-to-signal alignment (DTW). Whereas N1 and N2 are the lengths of the two sequences that need to be mapped, the original DTW has an $O(N_1N_2)$ time and memory complexity. Moreover, DTW has been used to analyze sequence data and is guaranteed to discover the best alignment between two series of time-ordered data [18].

Both approaches have flaws (base calling, DTW). The slowness of base calling is its biggest drawback, but the ONT team quickly increases the accuracy and efficiency of their base callers. The raw signal will be converted to read levels during base calling. A raw signal value will be converted to a nucleotide sequence (k-mers). Such a read-level operation might miss the unique details connected to an important genomic event, like an epigenetic event. Moreover, it has been found that base-calling error might increase significantly (10% to 25%) during the conversion of raw current signals to readings.

Because it directly compares the raw signal with the reference signal, the DTW method has a lower error. The time complexity of DTW is significantly larger, and this complexity (O(n2)) severely restricts the applications of DTW to various situations involving extremely long sequences. On the other hand, the translated predicted signal sequence and the nanopore's raw signal sequence are both extremely long. In reality, there is an order of magnitude disparity between the sampling rates of the raw signal sequence and the anticipated signal sequence since the frequency of electrical current measurements is 7-9 times higher than the passage speed of the DNA sequence. It is also difficult for DTW to resolve two input sequences with such a significant sampling rate and length disparity. [17]

We will next review some Nanopore Sequencing implementations and how they address the issues raised before.

1) Real-Time Mapping of Raw Signals with Uncalled: In nanopore sequencing, base callers convert raw signals to nucleotide sequences. But base calling is a computationally expensive task. It will require high computational power and time consumption. To overcome this, nanopore-selective sequencing is introduced. For that DTW algorithm is used rather than base calling. But the DTW algorithm has a quadratic complexity with gnome length. It will computationally intensive to use it for longer sequences. So using it is only efficient for sequencing the genome size of kilobase pairs.

Utility for Nanopore Current Alignment to Large Expanses of DNA (UNCALLED) (https://github.com/skovaka/UNCALLED) is proposed to address those drawbacks. Uncalled uses 'Read Until' to map raw signal streams to reference signals. And uses FM-index9 to search sequences in the reference DNAs. The FM-index algorithm checks for k-mers to represent the raw signal. It converts raw signals to events and calculates the possibilities of the event matches to k-mers using the ONT probabilistic model. To develop the FM-index search algorithm high-probability, k-mers are used, considering all possible sequences and locations. Then, the seed-clustering algorithm filters out false-positive locations by grouping seeds together. The raw signal of 100,000 E. coli reads was mapped to the E. coli K12 reference genome using a 3.0-GHz core to calculate the accuracy. With these, UNCALLED is more suitable for long sequence readings, eliminating the high computational demands in other methods [19].

2) Real-Time Mapping of Raw Signals with Sigmap: To overcome the drawback of the existing method of sequencing UNCALLED method was introduced. UNCALLED is efficient for gnome sizes ¡30Mbp and does not work properly for larger gnomes with a high repeat count. To address this issue, a new method is introduced called Sigmap. The sigmap method uses a 'Seed-and-Extend' strategy to map erroneous long signals. The 'Seed-and-Extend' strategy is used to find the exact or approximate alignments between signal readings and reference gnome signals and then matches the co-linear alignments and generates the final alignment. Sigmap also uses the same method. Another advantage of this method is that it can be used with noisy raw signals. This method involves FM-index9 for referencing and dividing raw signals into events. Then convert events into k-mers using the ONT pore model. This method keeps track of all the possible locations of every k-mers and removes the false-positive locations to avoid the noises in the raw signals. This is done by the seed clustering method [20].

Compared with the UNCALLED method, the Sigmap

approach achieves better success in sequencing small gnomes(<30 Mbp) than larger gnomes. As a result, it was given x4.4 speed and 11.49% more accuracy than the Uncalled method; not only that, it gave a higher F1 score. This research concludes that this approach can correctly map gnome size > 100Mbp.

3) Nanopore Selective Sequencing using Deep Learning: Deep Learning is a part of Machine Learning. It is based on Artificial Intelligence(AI) and Neural Networks. It is a trending topic nowadays because of its usage. Image processing, Natural Language Processing, Bioinformatics, Recommendation Algorithms, Deceive Identification, Advertising, and Fraud Detection are some use cases of Deep Learning. Analise RNA or DNA sequences using Deep Learning is another aspect of Selective Sequencing [20].

In this method, Nanopore Selective Sequencing is used with Deep Learning approach. The "ReadUntil" API comes with ONT and is used with Deep Learning scripts to perform the sequencing in real time. Here the first 200 nucleotides in a raw signal are used to classify the raw signals. Compared with the other available methods, some existing methods use raw signals only for the base calling, and then the raw signals are neglected. But here, raw signals can be used to accurately check the signals. Another approach is to use the DTW algorithm to selectively sequence the signals5. But this approach might have limitations in targeted and reference sequences. Another approach is that to overcome the limitations in the DTW algorithm, a base-caller is used to start sequencing and then use a reference mapping to select sequences needed to be sequenced. But it involves two steps(base calling and referencing) rather than a single step. Another approach is to use k-mers.

Mitochondrial DNA is classified as 'mitochondrial' or 'gnome' for the research. In this approach, error-prone base-calling is bypassed, and the deep learning modal can extract details from the raw signal in the process, increasing accuracy. One advantage of this approach is it neither nucleotide reference nor generated signal reference is not used to train the model. This will reduce the run time complexity of the model. The deep learning model categorises it as the 'mitochondria' or not before sending it through the sequencer. Because of that no need to sequence all the raw signals. Only required signals can be sequenced. This will prevent unwanted molecules are getting sequenced. This research shows that the Deep Learning approach can be used for real-time selective sequencing. It will give higher accuracy in analysing raw signals.

4) Readfish enables targeted nanopore sequencing of gigabase-sized genomes: Nanopore selective sequencing enables the selective sequence of specific gnomes by reversing the voltage. Dynamic Time Warping algorithm maps the raw signals to the reference. Since the DTW algorithm is high, computationally demanded GPU-based base calling is introduced in this research. To achieve this toolkit, 'Readfish'(https://github.com/looselab/readfish) is introduced. In this method, nucleotide sequences are used instead of raw signals. Because of that, converting raw signals to gnome references is unnecessary as the methods use DTW. This method concludes that base-calling using GPUs can be used for real-time nucleotide data steaming. This research provided the 'Readfish' toolkit for selective sequencing of gigabase-sized gnomes [21].

5) Nanopore Selective Sequencing using DTW: Unlike other traditional sequencing methods, ONT sequencing offers selective sequencing, which enables a selection of only a specific sequence of molecule sequences. It's a massive advantage over traditional sequencing methods, which preserve more resources and time. Sequence identification should be made with less delay to make the selection process more efficient. The DTW algorithm uses the 'Read Until' API of ONT minION to achieve selective sequencing in this method [5].

This concept matches the reading sequence directly to a reference sequence in real-time using the DTW algorithm. The algorithm matches two sequences while MinION manipulates the output sequence. In this approach, the 'Read Until' script (a computational tool. Not the 'Read Until' concept of ONT) selectively select the specific region from the sequence (The scripts are available to researchers freely under MIT licence: https://github.com/mattloose/RUscripts). Environment changes, voltage variations, noise, and interactions with other channels may cause disturbance in matching sequences. To measure, the disturbances are captured using three variables: shift, scale and drift. Z-score transformations can be used to overcome the shift and scale. Drift can be neglected in short sequences. So that using z-score transformations, external disturbances can be omitted.

III. Methodology

A. Introduction

This section will briefly introduce GPUs and why GPUs differ from CPUs. This will give the basic understanding that should follow through with the paper's next sections.

B. Graphics Processing Unit (GPU)

A GPU, or Graphics Processing Unit, is a specialised microprocessor designed to handle complex mathematical computations related to graphics processing, such as rendering images, videos, and animations. GPUs were initially developed for use in gaming and other graphics-intensive applications. Still, they have since found use in many industries, including machine learning, scientific computing, and cryptocurrency mining.

C. Graphics Processing Unit(GPU) vs Central Processing Unit(CPU)

The primary difference between a GPU and a CPU is that a GPU is built to carry out a variety of straightforward mathematical computations in parallel. A CPU, on the other hand, is built to carry out fewer, more sophisticated operations in a sequential fashion. As a result, jobs needing high levels of parallelism, including producing graphics or developing machine learning models, are well suited for GPUs. The memory architecture of GPUs and CPUs is another

difference. GPUs can access and process large amounts of data quickly because they typically have substantially more memory bandwidth than CPUs. This is crucial in applications that analyze streaming data in real time or use massive data collections. In conclusion, although both GPUs and CPUs are types of microprocessors, they are optimized for various tasks in different ways. While GPUs thrive in jobs requiring the parallel processing of massive amounts of data, particularly those connected to graphics processing and machine learning, CPUs excel at tasks requiring complex logic and decision-making.

D. GPUs' Advantages over CPUs

1) Parallelism: GPUs are substantially faster than CPUs for some tasks since they are made to process massive volumes of data in parallel. This is especially helpful in sectors like machine learning, where processing huge data sets quickly is required.

2) Memory Bandwidth: GPUs can access and process large amounts of data quickly because they typically have substantially more memory bandwidth than CPUs. This is crucial for applications that analyse real-time streaming data or use massive data collections.

3) Cost-effectiveness: In some cases, GPUs can be more economical than CPUs for certain computations. In data centres and other high-performance computing environments, for instance, GPUs can carry out some machine learning tasks quicker and with less power than CPUs, saving money.

4) Specialised Hardware: GPUs are designed with specialised hardware to accelerate certain types of computations, such as matrix operations, which are common in machine learning and other data-intensive applications.

5) Energy efficiency: GPUs can process data in parallel, making them more energy-efficient than CPUs for some tasks. As a result, data centres and other high-performance computing settings may benefit from lower energy costs and a smaller carbon footprint.

Ultimately, depending on the exact use case and the kind of computations being done, GPUs may be preferable to CPUs in some situations. GPUs can offer considerable performance and cost advantages over CPUs for data-intensive applications, including machine learning.

E. Types of GPUs

Some of the most common types of GPUs are as follows:

1) Integrated GPUs: These GPUs, generally found in consumer-level laptops and desktops, are built into the CPU. Web browsing and video playback are just a couple of the core graphics processing activities that they are made to handle.

2) Dedicated GPUs: These standalone GPUs are for more demanding graphics processing applications like gaming, video editing, and 3D rendering. Dedicated GPUs range from entry-level models to top-of-the-line gaming and workstation GPUs, with prices and performance levels to match.

3) Datacenter GPUs: These are high-performance GPUs for large-scale computing environments like data centres. They are optimized for data-intensive applications like machine learning, scientific computing, etc.

4) Mobile GPUs: These are designed for mobile devices like smartphones and tablets. They are optimised for low power consumption and are typically less powerful than dedicated or data centre GPUs.

5) Workstation GPUs: These are high-performance GPUs designed for professional workstations such as 3D modelling, animation, and scientific visualisation.

6) Cloud GPUs: These GPUs are accessible via cloud computing services, giving consumers instant access to high-performance GPU computing resources. Machine learning, scientific computing, and other data-intensive applications needing high parallel processing capabilities benefit greatly from cloud GPUs.

F. NVIDIA GPUs

NVIDIA Corporation, a top producer of GPUs and other computing hardware, created NVIDIA GPUs, a graphics processing unit. Complex graphics processing activities, as well as other high-performance computing workloads like machine learning and scientific computing, can be handled by NVIDIA GPUs.

Various product lines offer NVIDIA GPUs, from entry-level consumer GPUs to high-performance data centre GPUs. The following are some of the most well-known NVIDIA GPU product lines:

1) GeForce: These are NVIDIA's consumer-level GPUs for graphics-intensive software like games. GeForce GPUs come in various performance configurations, from entry-level to top-of-the-line gaming GPUs.

2) Quadro: These are workstation-grade GPUs from NVIDIA intended for use in professional workstations for activities including 3D modelling, animation, and scientific visualization. Quadro GPUs have capabilities like ECC memory and support for multiple screens designed for great performance and dependability.

3) Tesla: These NVIDIA GPUs are intended for large-scale environments like machine learning and scientific computing. These are data centre-level GPUs. Tesla GPUs are equipped with features like NVIDIA's Tensor Core technology for accelerating machine learning workloads and are optimized for high performance and energy economy.

High performance, dependability, and compatibility with software tools and frameworks in industries like machine learning and scientific computing are hallmarks of NVIDIA GPUs. NVIDIA GPUs are used by several of the most potent supercomputers in the world to speed up workloads. NVIDIA GPUs are well-liked for many high-performance graphics processing applications, ranging from gaming and entertainment to machine learning and scholarly research.

G. CUDA Programming

NVIDIA created the parallel computing platform and programming style known as CUDA (Compute Unified Device

Architecture) for their GPUs. It enables programmers to speed up computationally demanding applications like machine learning, scientific simulations, and video processing by utilizing the parallel processing capabilities of NVIDIA GPUs. A programming language similar to C and a collection of APIs (Application Programming Interfaces) is used in CUDA programming to allow programmers to create parallel programs that can run on NVIDIA GPUs. The kernel notion, which refers to compact, heavily parallelized functions that can run concurrently on the GPU, serves as the foundation for the programming model.

To use CUDA programming, developers need an NVIDIA GPU and the CUDA Toolkit, which includes the CUDA programming language, APIs, and compiler. The CUDA programming model enables programmers to create programs that can utilize contemporary GPUs' enormous parallel processing capacity, allowing them to perform computations much more quickly than conventional CPUs. Applications for CUDA programming range from machine learning and video processing to data analytics and scientific simulations. It has become a popular tool for programmers who want to speed up their programs and improve performance, especially in industries that demand a lot of parallel processing capacity.

IV. Experiment Setup and Implementation

A. Introduction

This section will describe GPUs and CUDA programming in our research project. We will discuss our approach to optimising the performance of our code on the GPU, including strategies for data management, thread synchronisation, and memory access. Overall, this section will provide a comprehensive guide to our methodology for leveraging the power of GPUs and CUDA programming to accelerate the Dynamic Time Warping Algorithm for Nanopore Selective Sequencing.

B. CUDA Programming Steps

You ought to better understand the fundamental phases in GPU programming using CUDA before moving on to the implementation specifics. They are listed below:

1) Allocating memory: The first step in CUDA programming is allocating memory on the GPU using the CUDA runtime API. This entails building device memory pointers that can be used to send data between the CPU and GPU.

2) Transfer data: Transferring data from the CPU to the GPU is the next step after allocating memory to the GPU. The CUDA runtime API functions cudaMemcpy or cudaMemcpyAsync can be used for this.

3) Tesla: Launch kernels: Kernels are quick, highly parallelized operations that can run simultaneously on the GPU. Programmers must first define the kernel function and its execution parameters, including the number of threads and blocks to be launched, before they may launch a kernel in CUDA.

4) Data processing: After the kernel has been started, each thread concurrently runs the kernel function on a distinct section of the data. The kernel code can be optimized for maximum efficiency using CUDA-specific features like shared memory and atomic operations.

5) Transfer results: Results must be returned to the CPU for additional processing or storage when the kernel has finished processing the data. Like the data transmission stage, this can be accomplished using the cudaMemcpy or cudaMemcpyAsync routines.

6) Free memory: Using the cudaFree function of the CUDA runtime API, the programmer must finally free the memory allocated to the GPU.

These actions make up the fundamental structure for CUDA GPU programming. The programmer must carefully plan and optimize the kernel functions for parallel processing to execute certain algorithms or applications, considering elements like data dependencies, thread synchronization, and memory access patterns.

C. Existing Code

The existing code base was written in C language, which could extract the signal data from standard file format, preprocess them and compare them with the reference signal. Existing code ran in the CPU, and no parallel programming techniques were used. In the code base, the whole process was divided into several parts. Those were read the signal data from the standardised file format. The next program would normalise the signal data (preprocessing) to match it with reference data and check its validity. After that signal would be converted into a set of floating point numbers. Also, if the reference signals were a sequence of nucleotides, they would be converted into a signal domain because both read data and reference should be in the same domain. Next, reference signals would be matched with read signals. This step divides read signals into smaller parts, and the subsequence dtw score is calculated to identify the optimal match. In the whole process, this would consume most of the processing time and computing power, and also, in this project, we were only considering this part. Since this part took most of the processing time, we identified the response functions for that, and our goal was to reduce the processing time.

In the comparison process, there were a few biological details to consider, which were used in the comparing process. In DNA sequencing, we needed to compare the entire genome, but the entire genome had a huge amount of details and could not compare the entire genome directly. In that case, reference DNA genomes were separated into multiple chromosomes. If the genome is a book, chromosomes are like chapters. In the human genome, there are 23 chromosomes, and some viruses only have one chromosome. Since reference signals came as chromosomes, read data was needed to compare with each chromosome. It was decided that the optimal matching chromosome was the correct match for the given read.

When nanopore sequencers fetched the reads, it also came as multiple reads for the single genome. That is because

genomes consisted of chromosomes, and those were very fragile. When the DNA came as a solution, it consisted of pieces of each chromosome. So sequencer randomly picked a sample which would be a part of any chromosome. Like this, it had several readings, each with a long sequence of floating point numbers. In practice, the first part of the read would remove (trimmed) because it was of no use; most of the time, it was inaccurate. Next, extract a portion of the read signal from each read (for example, 3000 samples from the read signal). Comparing a whole read with reference would take unnecessary processing time, and the DTW algorithm did not give good results for the longer sequences. Also, each read-only needed to know which chromosome matches the most, and for that portion of a read signal is enough and no need for all the details in the read signal. After that portion of the reading would compare with all the chromosomes of the reference signal and repeat that for all the reading signals. A portion of the read signal was compared with each chromosome, and calculated the DTW score and found in the chromosome with a minimum score with the given portion, and it would decide that the read belonged to that chromosome. In the existing code, they used several nested for loops to achieve this process, which means there was potential to parallelise the process and reduce the running time.

D. Proposed Method

As previously mentioned, the project only considered the part where the reference signal compares with the read signal. It was the most time-consuming part, and since there are a lot of nested loops in that function, there is a potential to optimise it using parallel computing techniques. Also, the algorithm-wise DTW is more of a sequentially running algorithm, and calculations depend on the previous values. Because of that, it isn't easy to parallel do the calculations in DTW. In the research, we found effective solutions and modifications for the algorithm, which increased the degree of the parallelism other researchers proposed.

The growth of GPU performance in parallel processing has been significant over the past few decades, with several key advancements and innovations leading to a dramatic increase in computational power.

- CUDA: In 2007, NVIDIA introduced CUDA, a parallel computing platform that allows developers to use NVIDIA GPUs for general-purpose computing. CUDA provides a programming model enabling developers to write GPU code using familiar programming languages like C, C++, and Python.
- GPGPU: General-purpose computing on graphics processing units (GPGPU) is a technique that uses the parallel processing power of GPUs for non-graphics applications. This technique has been around since the early 2000s but has gained popularity recently due to the increase in GPU performance.

Overall, the growth of GPU performance in parallel processing has been driven by a combination of hardware and software advancements, including the introduction of programmable GPUs, GPGPU, CUDA, Tensor Cores, and dedicated ray tracing hardware. These innovations have greatly expanded the possibilities for high-performance computing. We planned to divide the calculation parallelly using GPU's computational power in the proposed way.

As previously mentioned, parallelising the DTW algorithm could reduce the processing time. The dynamic programming technique, on which the dynamic time warping algorithm is based, saves the previous computations' results in a matrix. The majority of matrix calculations can be parallelised to perform computations more quickly. Also, since DTW does not require sophisticated mathematical models, it is simple to divide the procedure into smaller calculations and boost efficiency. The computation of each iteration in the matrix depends on the outcome of the previous step, which restricts the algorithm's capacity to be parallelised. The degree of parallel ability in DTW can be increased in several ways. The cost matrix of the DTW represents the DTW distance between two data points in each cell. Each cell's calculation is based on the outcomes of the left, top, and top-left cells. For cell (i,j) in the matrix D, the value of D[i, j] depends on the values in D[i, j-1], D[i-1, j], D[i-1, j-1]. In that case, for any cell (i,j) in the matrix, all cells between (0,0) and (i,j) need to be computed before calculating the value of the position (i,j). This computation is carried out by the DTW method in either a row- or column order. Sequential computations are made for the cells in the same row or column. However, the diagonal cells don't share any info, which is the opposite of the former. As a result, simultaneous computations for diagonal sets of cells are possible. There is a data dependency between two adjacent diagonals. As a result, we could only calculate the data from one diagonal set at a time. The calculation is performed on the first cell in the closest diagonal line set and then updates the remaining cells. These diagonally positioned cells can be computed concurrently because they only depend on the first cell. After completing the diagonal set, proceed consecutively to the next one. Using this approach, data dependency is ensured, and reasonable parallelism can be attained.

E. GPU Implementation

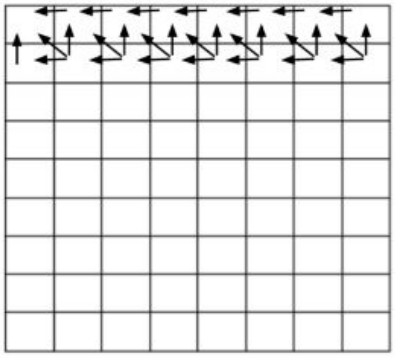
As discussed before, GPU implementation consists of several stages. We will discuss each step and implementation details of implementing the DTW algorithm for GPU for Nanopore Selective Sequencing.

1) Memory Allocation: The cudaMalloc() function is one of many the CUDA run-time API provides to allocate memory on the hardware. The cudaMalloc() function returns a reference to the allocated memory after allocating a block of device memory of the requested size (in bytes).

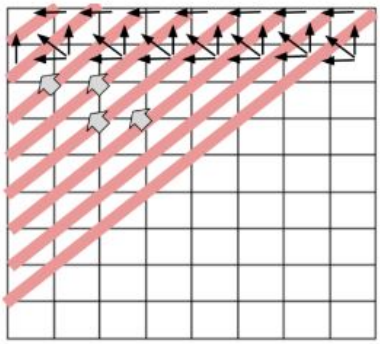
cudaMalloc(void**Ptr, sizet size)

The first argument, Ptr, is a pointer to the device memory address that will be allocated. The second argument, size, is the number of bytes to allocate.

Data in the existing code consisted of complex structs. GPU can not access those data directly from the CPU memory.



(a) Computation dependencies



(b) Diagonal elements of the cost matrix

Fig. 2: [22]

Because of that, data have to be converted to a set of data arrays. Those arrays will be copied to the GPU memory in the next step. Before cudaMalloc(), all the complex arrays should be converted to simple linear arrays. After that, for those arrays, CUDA memory will be allocated. Following is an example of coping to linear arrays.

len_raw_signal[i] $<--$ db $->$ slow5_rec[i] $->$ len_raw_signal;
et_n[i] $<--$ db $->$ et[i].n;
qstart[i] $<--$ db $->$ qstart[i];
qend[i] $<--$ db $->$ qend[i];
qlen[i] $<--$ qend[i] - qstart[i];

2) Transfering Data: The CUDA runtime API's cudaMemcpy() function frequently copies data between host and device memory.

cudaMemcpy(void* dst, const void* src, size_t count, cudaMemcpyKind kind)

The first argument, dst, is a pointer to the destination memory where data will be copied. The second argument, src, is a pointer to the source memory from where data will be copied. The third argument, count, specifies the number of bytes to be copied. The fourth argument, kind, specifies the direction of the copy and can take one of the following values: cudaMemcpyHostToDevice: Copies data from the host memory to the device memory. cudaMemcpyDeviceToHost: Copies data from the device memory to the host memory.

To copy data from the host to the device, you can use the cudaMemcpy() function with cudaMemcpyHostToDevice as the fourth argument. In this step, all the arrays are copied to the GPU memory, and now GPU can access those arrays and values.

3) Data Processing: Once the above steps are completed, all the pointers to the arrays are passed to the CUDA kernel function, as shown below. It will execute the CUDA kernel and the DTW algorithm in the GPU.

dtw_single2<<<NUMBER_of_BLOCKS, SIZE_of_BLOCK>>>(args...)

In the GPU implementation, "SIZE_of_BLOCK" is can be changed as the requirements. With that change, "NUMBER_of_BLOCKS" will be changed. There are several possibilities.

- One BLOCK and the number of signals as the "SIZE_of_BLOCK."
- The number of signals as BLOCK clout and one signal per BLOCK.
- Variant amount of BLOCKs and signals per BLOCK

4) Transfering Results: cudaMemcpy() function is used to copy the result into the CPU memory with the fourth argument as cudaMemcpyDeviceToHost.

5) Freeing memory: The cudaFree() function is used to free memory previously allocated on the device using functions like cudaMalloc().

cudaFree(void* Ptr);

The only argument is a pointer to the memory block to be freed. This function deallocates the memory block previously allocated with cudaMalloc()

V. Results and Analysis

This section demonstrates the DTW performances for Nanopore selective sequencing for different optimisation strategies with various hardware configurations. We used the following hardware platforms.

Platform 1:
GPU: NVIDIA GeForce MX330 with 384 cores
Platform 2:
GPU: NVIDIA Tesla K40 with 2880 cores

The CPU code described in the section on existing code runs exclusively in the CPU sequentially. It does not use a parallel mechanism. The following table illustrates how long it takes to calculate DTW on each platform.

	Time(s)
Platform 1	30
Platform 2	23

The time shown above is the total time DTW has spent on all reads. Our ultimate objective was to lower this time

by using various optimisation approaches. Thus we preserved those times as the reference times for each platform.

As described in the methodology section, we transformed the DTW computation phase from C code to CUDA. At first, we just used one block for the entire data set's reads. The data set contains 485 readings, and 485 concurrent threads were used to match each read with the reference signal. The parallelisation did not improve performance when compared to CPU code. The time required by each of the aforementioned platforms is shown in the table below.

	Time(s)
Platform 1	256
Platform 2	981

Compared to the CPU code time, we can see that the time has significantly increased. Although the process had been parallelised, the GPU used only one core to perform the program because we utilised only one block. CPU cores are more powerful than GPU cores. As a result, we failed to improve performance by utilising the aforementioned strategy.

Then, for each read, we used one block. As part of our second technique, we parallelised the operation and utilised 485 blocks. As compared to the first technique, it demonstrated a considerable improvement, but not when compared to the CPU code.

	Time(s)
Platform 1	218
Platform 2	158

We used one block for each read, which is what caused the improvement to be so noticeable when compared to the first technique. As a result, those blocks are dispersed across numerous GPU cores, yet sequential matching of each read with the reference signal continues to run inside of a single thread. With our third optimising technique, we attempted to parallel that action. The third strategy's calculation time is displayed in the table below.

	Time(s)
Platform 1	281
Platform 2	252

When compared to the second technique, there was no improvement with this one. Performance was not improved by invoking one CUDA kernel inside another CUDA kernel.

VI. Future Work

We have displayed several findings using various optimisation procedures in the results section. The best optimised times we obtained for platforms 1 and 2 were 218 and 158 seconds, respectively. The outcome was different from what we had previously anticipated.

As part of future work, the DTW calculation segment will be parallelised. Most of the time-consuming activities take place there. As a result, we will give priority to paralysing this section. To calculate the DTW matrix, we'll use the diagonal approach.

References

[1] Senin, P. (2008). Dynamic time warping algorithm review. Information and Computer Science Department University of Hawaii at Manoa Honolulu, USA, 855(1-23), 40.

[2] M. Muller, M. (2007). Dynamic time warping. Information retrieval for ¨ music and motion, 69-84.

[3] Lahreche, A., Boucheham, B. A Comparison Study of Dynamic Time Warping's Variants for Time Series Classification. International Journal of Informatics and Applied Mathematics, 4(1), 56-71.

[4] Poli, G., Mari, J. F., Saito, J. H., Levada, A. L. (2007, October). Voice command recognition with dynamic time warping (dtw) using graphics processing units (gpu) with compute unified device architecture (cuda). In 19th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'07) (pp. 19-25). IEEE

[5] Permanasari, Y., Harahap, E. H., Ali, E. P. (2019, November). Speech recognition using dynamic time warping (DTW). In Journal of Physics: Conference Series (Vol. 1366, No. 1, p. 012091). IOP Publishing

[6] Bagnall, A., Lines, J., Vickers, W., Keogh, E. (2018). The UEA UCR time series classification repository. URL http://www. timeseriesclassification. com, 122.

[7] Ding, H., Trajcevski, G., Scheuermann, P., Wang, X., Keogh, E. (2008). Querying and mining of time series data: experimental comparison of representations and distance measures. Proceedings of the VLDB Endowment, 1(2), 1542-1552.

[8] Dau, H. A., Silva, D. F., Petitjean, F., Forestier, G., Bagnall, A., Mueen, A., Keogh, E. (2018). Optimizing dynamic time warping's window width for time series data mining applications. Data mining and knowledge discovery, 32(4), 1074-1120.

[9] Shanker, A. P., Rajagopalan, A. N. (2007). Off-line signature verification using DTW. Pattern recognition letters, 28(12), 1407-1414

[10] Sakoe, H., Chiba, S. (1978). Dynamic programming algorithm optimization for spoken word recognition. IEEE transactions on acoustics, speech, and signal processing, 26(1), 43-49.

[11] Wang, Y., Zhao, Y., Bollas, A., Wang, Y., Au, K. F. (2021). Nanopore sequencing technology, bioinformatics and applications. Nature biotechnology, 39(11), 1348-1365.

[12] Deamer, D., Akeson, M., Branton, D. (2016). Three decades of nanopore sequencing. Nature biotechnology, 34(5), 518-524.

[13] Leggett, R. M., Clark, M. D. (2017). A world of opportunities with nanopore sequencing. Journal of experimental botany, 68(20), 5419-5429.

[14] Deamer, D., Akeson, M., Branton, D. (2016). Three decades of nanopore sequencing. Nature biotechnology, 34(5), 518-524.

[15] Danilevsky, A., Polsky, A. L., Shomron, N. (2021). Real-time selective sequencing using nanopores and deep learning.

[16] Payne, A., Holmes, N., Clarke, T., Munro, R., Debebe, B. J., Loose, M. (2021). Readfish enables targeted nanopore sequencing of gigabasesized genomes. Nature biotechnology, 39(4), 442-450.

[17] Loose, M., Malla, S.,Stout, M. (2016). Real-time selective sequencing using nanopore technology. Nature methods, 13(9), 751-754

[18] Han, R., Li, Y., Gao, X., Wang, S. (2018). An accurate and rapid continuous wavelet dynamic time warping algorithm for end-to-end mapping in ultra-long nanopore sequencing. Bioinformatics, 34(17), i722-i731.

[19] Kovaka, S., Fan, Y., Ni, B., Timp, W., Schatz, M. C. (2021). Targeted nanopore sequencing by real-time mapping of raw electrical signal with UNCALLED. Nature biotechnology, 39(4), 431-441.

[20] Zhang, H., Li, H., Jain, C., Cheng, H., Au, K. F., Li, H., Aluru, S. (2021). Real-time mapping of nanopore raw signals. Bioinformatics, 37(Supplement1), i477-i483

[21] Payne, A., Holmes, N., Clarke, T., Munro, R., Debebe, B. J., Loose, M. (2021). Readfish enables targeted nanopore sequencing of gigabasesized genomes. Nature biotechnology, 39(4), 442-450

[22] Zhu H, Gu Z, Zhao H, Chen K, Li C, He L (2018). Developing a pattern discovery method in time series data and its GPU acceleration, 1(4), 266-283.