A Software Framework for the Detection and Classification of Biological Targets in Bio-Nano Sensing

Abdul Hafeez

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in the partial fulfillment of the requirement for the degree of

Doctor of Philosophy
In
Computer Science and Applications

Ali R. Butt, Chair
Samir M. Iqbal
B. Aditya Prakash
M. Mustafa Rafique
Eli Tilevich

July 23, 2014
Blacksburg, Virginia, USA

Keywords: Automated Detection, Parallel and Distributed Computing, Solid-state Micropores, Accelerated Diagnosis

A Software Framework for the Detection and Classification of Biological Targetss in Bio-Nano Sensing

Abdul Hafeez

ABSTRACT

Detection and identification of important biological targets, such as DNA, proteins, and diseased human cells are crucial for early diagnosis and prognosis. The key to discriminate healthy cells from the diseased cells is the biophysical properties that differ radically. Micro and nanosystems, such as solid-state micropores and nanopores can measure and translate these properties of biological targets into electrical spikes to decode useful insights. Nonetheless, such approaches result in sizable data streams that are often plagued with inherit noise and baseline wanders. Moreover, the extant detection approaches are tedious, time-consuming, and error-prone, and there is no error-resilient software that can analyze large data sets instantly. The ability to effectively process and detect biological targets in larger data sets lie in the automated and accelerated data processing strategies using state-of-the-art distributed computing systems.

In this dissertation, we design and develop techniques for the detection and classification of biological targets and a distributed detection framework to support data processing from multiple bio-nano devices. In a distributed setup, the collected raw data stream on a server node is split into data segments and distributed across the participating worker nodes. Each node reduces noise in the assigned data segment using moving-average filtering, and detects the electric spikes by comparing them against a statistical threshold (based on the mean and standard deviation of the data), in a Single Program Multiple Data (SPMD) style. Our proposed framework enables the detection of cancer cells in a mixture of cancer cells, red blood cells (RBCs), and white blood cells (WBCs), and achieves a maximum speedup of 6X over a single-node machine by processing 10 gigabytes of raw data using an 8-node cluster in less than a minute, which will otherwise take hours using manual analysis.

Diseases such as cancer can be mitigated, if detected and treated at an early stage. Micro and nanoscale devices, such as micropores and nanopores, enable the translocation of biological targets at finer granularity. These devices are tiny orifices in silicon-based membranes, and the output is a current signal, measured in nanoamperes. Solid-state micropore is capable of electrically measuring the biophysical properties of human cells, when a blood sample is passed through it. The passage of cells via such pores results in an interesting pattern (pulse) in the baseline current, which can be measured at a very high rate, such as 500,000 samples per second, and even higher resolution. The pulse is essentially a sequence of temporal data samples that abruptly falls below and then reverts back to a normal baseline with an acceptable predefined time interval, i.e., pulse width. The pulse features, such as width and amplitude, correspond to the translocation behavior and the extent to which the pore is blocked, under a constant potential. These features are crucial in discriminating the diseased cells from healthy cells, such as identifying cancer cells in a mixture of cells.

Dedication

Dedicated to my parents,
for their endless love, encouragement, prayers, and support …

ACKNOWLEDGEMENTS

One never notices what has been done; one can only see what remains to be done.
- Marie Curie

I would like to express my deepest gratitude to my advisor, Ali R. Butt, for his perpetual guidance, valuable advice, and everlasting encouragement during my graduate studies at Virginia Tech. He has always been a continuous source of help, motivation, and an inspiration for me. Those were unforgettable events when he used to stay up all night with us to meet paper deadlines. He has always been passionate, available there to help, provide useful and insightful advices, and feedback on papers, presentations, and even, pushing and supporting us to attend conferences and interact with the experts in the field in order to gain an up-to-date knowledge of state of the art. Without his continuous support, I couldn't have completed this journey, and this work could have been a daunting task.

I would like to thank my advisory committee members, Samir M. Iqbal, B. Aditya Prakash, M. Mustafa Rafique, and Eli Tilevich for their helpful and invaluable feedback on my graduate research. Particularly, I would thank Dr. Samir Iqbal for giving me an opportunity to explore the use of Computer Science in the emerging field of bio-nanotechnology. It was this first collaboration that sparked further ideas and opportunities for me and laid down the foundation of my dissertation topic. He was always kind, passionate, and helpful. I would like to thank my neighbor - Eli for enriching my PhD experience. He has a remarkable pleasant and easy-going personality. His unique sense of humor helped in my graduate studies and made the time spent in graduate school a real fun. He taught us how to communicate effectively and efficiently. I found Eli an outstanding writer who has strong presentation skills which are crucial to computer science research. Overall, his encouragement, invaluable advice, and appreciation of my work boosted my confidence level and help me accomplish my milestones throughout my PhD.

I would like to thank M. Mustafa Rafique for his support and encouragement. Despite his busy schedule at IBM Research, he always managed to provide quick and prompt responses to my emails. I very appreciate the skype brainstorming sessions and technical discussions. His ability to manage time, quick feedback through email and skype, and his work ethic are always inspiring.

I am also deeply honored and fortunate to have Aditya Prakash as my program committee member. I would like to thank him for taking his time out of his busy schedule in order to provide me with invaluable feedback during my prelim and final defense.

I am grateful to Lynn Abbott for his availability to discuss pattern recognition and machine learning algorithms. His invaluable feedback and encouragement increased my confidence to explore and adapt machine learning to my research. Thanks are due to Waseem Asghar for his help to understand and analyze the data collected from bio-nano sensing. Thanks to Anne Ryan in explaining the concepts of statistics in research, which I later applied to the data analysis in my research.

I'd like to express my acknowledgement to IBM researcher Prasenjit Sarkar for insightful guidance of project defining, productive advices, and energetic conversations on studying and exploring the use of large-scale distributed systems in big data problems, especially, accelerating protein simulations.

I would like to thank my colleagues at Distributed Systems and Storage Lab (DSSL) - Henry Monti, Dr. Min Li, Aleksandr Khasymski (Alex), and Hyogi Sim for their constructive feedback and technical discussions. Particularly, I will thank Alex for his useful comments and though-provoking discussions on graphics processing units (GPUs) and access to GPU cluster machine. Henry and Min were such a passionate and critical audience, who would share their wisdom, and receptive enough that I could try my research ideas on, and were always available for instant feedback on my work and long discussions.

I'd like to thank Hyogi Sim, Yue Cheng, Krish K. R, Tariq Kamal, Ali Anwar, and M. Safdar Iqbal for all the wonderful discussions we had while hanging out in the lab and during the coffee and

lunch breaks. I can’t imagine such a better bunch of people as grad school colleagues. Thanks to them, I am a better-rounded thinker than I used to be.

I’d like to thank Sharon Kinder-Potter, Ginger Clayton, and Jessie Eaves for making all the administrative matters certainly easier to deal with. Thanks are due to Rob Hunter and Ryan Chase for providing me access to the cluster machine and handling several IT issues for me. I deeply appreciate their help.

I’d also thank all my friends in CS@VT and VT in general for all the fun we had together during the entertaining events and fantastic moments.

Many special thanks to my sisters, brothers, and brother-in-law for their passionate love and endless support towards me. They have always made me feel like a special one and helped me in any possible way.

Finally, I would like to thank my beloved parents for the immeasurable, unconditional love, encouragement and everlasting support since childhood. I probably would not be writing this dissertation if it weren’t for my parents who have provided me with the foundations of a good education. Their endless persistence, determination and unshakable faith have always been a motivation and a driving force to endure and survive in any and all challenging circumstances during my studies. Now, when I am the only PhD in my family, I consider this work to be as much an achievement of theirs as mine.

I pray to Almighty Allah, who bestowed all these blessings on me, to give me the strength and wisdom to use this knowledge the way He wants.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1. Problem Statement

The problem with the traditional technologies, including fMRI (function Magnetic Resonance Imaging), Computerized Tomography, and biopsy is that they can detect cancer at later stages. Miniaturization, coupled with omics revolution, has made robust interfaces possible to faithfully measure and transduce important biological interactions at fundamental levels. These endeavors provide tools to diagnose diseases at early stages, monitor prognosis in higher details with minimally invasive procedures, and follow the trends of disease progression in response to new drugs or novel therapies. However, a multi-faceted challenge shared to these advancements is that these suffer from the noisy raw data which is generated at high rates. Moreover, the processing and post-processing often entail manual analysis, which is inefficient and error-prone. To our knowledge, there is no automated and error-resilient software that can detect the biological targets in real time. This possibly incurs unacceptable delays in the detection of medical conditions and diseases. In simple terms, the amount of measured data per unit time has grown exponentially both in resolution and dimensions. Therefore, it has become near impossible to extract useful patterns from the data in real time.

1.1.1. Bio-Nano Sensing

The technologies developed at a micro- and nano-scale are inherently high performing, and the data collected from these interfaces is often at micro- and nano-scale – generating millions of samples within a fraction of a second. Solid-state nanopore and micropore are examples of such devices that provide interfaces to translocate biological targets, such as DNA, and human cells through it. The translocation phenomenon of these targets through the pore blocks the output current partially that results in an

interesting pattern in the output. Unfortunately, the data is generated at high rates and riddled with noise and baseline artifacts. Furthermore, a typical blood sample which is a fraction of milliliter blood sample, when translocate through the solid-state micropore result in few GBs of data. In state-of-the-art, the performance and speed are hardly ever discussed for these devices with respect to the computing resources employed for measurements and analysis.

The accurate detection and processing of interesting patterns are also critical in the analysis of the biological targets for diagnostics and therapy, otherwise, results into false alarms. This leads to degraded sensitivity and poor selectivity of the system, and therefore, loss of system integrity and reduced usefulness of miniaturization.

Another key challenge that is common to bio-nano sensing is the unavailabitily of a software that can provide pattern-detection in real-time. Unfortunately, the collected data has to be processed and analyzed manually. Manual analysis is time-consuming, tedious and error-prone, and is subject to technician's experience. The solid-state nanopores that can sample up to 454,000 samples a second need backend software to record the biological events such as DNA, proteins and disease biomarkers faithfully at higher resolution and discretization. Unfortunately, the acquired data is complicated by large background noise and baseline artifacts in the measured data that poses challenge towards the accurate detection, and makes the manual detection process even complicated. Furthermore, applications like EEG, ECG, and MRI suffer from large amount of raw data where an analyst has to wait until a significant amount of data is accumulated, and then process it for the detection of biological targets, and further analysis. Thus, the useful decision-making never happens in real-time due to the offline collection of raw data, and the manual interventions performed on the said data.

1.1.2. Challenges in Bio-Nano Sensing

The identification and analysis of patterns are fundamental to many applications. Typically, the patterns can be upward spikes forming peaks, or inverted peaks forming troughs. In either case, closer examination of peaks leads to useful insights relevant to the domain. In power distribution data, peaks indicate the trend pertaining to high demands. In server CPU utilization data, peaks indicate trends in the workload characterization. In network data, a peak corresponds to burst in the traffic. In financial data, peaks show sharp rises in price or volume. Medical research and modern clinical setup depict an increase in recording important physiological signals, which substantially involves the detection of peaks/troughs in the target signals. Examples include patterns in the electrocardiogram [1], electroencephalogram [2], neuronal activity in experimental animals [3, 4], peak inspiration and end expiration[5], patterns in hormone secretion [6, 7], counting smooth muscle contractions [8], and the spectral peaks in chromatography [9]. Additional applications such as mass spectrometry [10, 11], bioinformatics [12], astrophysics [13, 14], signal processing [15], image processing [16], and other related domains [17-20] – entail peak detection techniques.

However, the process of peak-detection suffers from a couple of challenges. The major challenge is the arbitrary definition of the patterns itself. Part of the reason is the tight coupling of the definition to the application domain, instead of having one single universal interpretation. It can be an abrupt rise followed by a sudden fall in time, such as spikes in musical note. It can be a sharp rise followed by a gradual pattern spread, or vice versa, while monitoring the price-level of oil to capture the recession period. The patterns resulting from the human respiratory system have progressive rise and fall at either end with the shape of the peak could be symmetric, or asymmetric. There can be an abrupt rise/fall at one end, while gradual transition at the other end. The task becomes even more challenging when the patterns are aperiodic, furthermore, they are occurring at different scales, and amplitudes. Finally, the true peaks that capture the useful insights are often hidden in the noise. Thus, it is very daunting to agree on a single definition of a pattern. Nevertheless, the existence of a wide range of applications that incorporate

patterns suffer from collective challenges, though each application has evolved into their idiosyncratic definition, and therefore, their peculiar pattern-detection algorithms. The communal challenges are summarized as below:

Noise:

The data is often riddled with baseline wander, and noise from the source. Therefore, it is necessary to estimate the right level of noise in order to avoid the detection of false alarms, which can adversely effect precision and recall of the system.

Criterion to set the threshold:

A static threshold to detect peaks/troughs won't work when there are baseline shifts. This can prevent the detection of true peaks. In contrast, an adjustable threshold will compensate for static threshold. However, the criterion to setup such an adaptive threshold is very crucial in order to acheive high precision.

Patterns at different scale/amplitude:

This problem arises due to the patterns that occur at varying scales and magnitudes that make it challenging to differentiate the strong and weak pulse. This problem exists even after the patterns have been detected in the data against the selected threshold. However, in some case, both types of patterns are important. For instance, in bio-nano sensing, different types of phenomenon is captured by different patterns in the underlying process.

Tunable parameters:

Finally, the existence of highly unpredictable data result in tuning of different parameters of the devised algorithm. Such tuning of the algorithms based on the arising situation is unmanageable and entails subjectivity, which is time-consuming and error-prone. Therefore, devising self-tunable algorithms in order to minimize the tunable parameters is pivotal for pattern detection.

Large-scale data:

The resulting noisy data from bio-nano sensors is high-throughput data. Data generated even by a single sensor is at 2MB per second. Multiple sensors launched in a typical clinical setup measuring biological targets in parallel will easily become a big data problem.

1.2. Motivation

Bio-nanotechnology is promising at a scale varying from human cells down to the nano-scale, such as proteins and DNA. Micro- and nano-scale devices enable early detection of diseases, such as cancer, and sense biological targets at ever finer granularity [21-29]. However, the problem with such techniques is that they suffer from noisy data deluge, and secondly; the detection process is done manually, which is too inefficient. Atop, passing a biological target, such as *DNA*, through a solid-state nanopore can result in as many as 200,000–500,000 distinct values per second [30-32]. Such as data are beyond the capabilities of an individual computer machine, especially for real time processing [30]. Equally important, a well-trained technician has to spend innumerable hours to extract useful patterns from few GBs of raw data – collected from the translocation a blood sample, which is in few milliliters. This is non-trivial and entails handling of huge amounts of data. Moreover, the data-deluge that is thus created – by measurements (nanopores) where thousands of sensors generate lots of data – requires innovative solutions for efficient computing and analysis, and a cost-efficient solution in this design-space may preclude the use of traditional computing resources.

Pattern-detection is of the utmost importance in disease diagnosis technologies, such as mass spectroscopy, ECG, and MRI, to name a few [33, 34]. With the advent of novel biological application of solid-state micro- and nano-scale devices [27, 35-37], the problem of pattern-recognition is now coupled with the enormity of the datasets [38]. In temporal measurements, it is easy to visually analyze the patterns and pulses when the collection time scales are minuscule and the measured events are plenty [34]. However, when the collected data is huge and the actual events or pulses constitute a tiny fraction of the acquired data, the automation of the event recognition becomes indispensable. Solid-state micropores

are small orifices in silicon-based membranes that have been used to electrically measure the passage of human cells through these [23]. When no cells are present, the ionic current is measured temporally, which gives a stable baseline. As soon as a cell passes through, a dip in the baseline is measured that stems from the blockage of the micropore. The translocation events are thus registered as pulses and the features of the pulses depict the pattern specific to the cell type [39].

Furthermore, the pattern-detection in the data is done manually, which is monotonous, time-consuming and error-prone – thus, it is critical to automate the data analysis for efficient, and accurate detection of the patterns. However, this is non-trivial and entails handling of the huge amount of data as well as significant computing resources. This requires innovative solutions for efficient computing and analysis, and a cost-efficient solution in this design-space may preclude the use of traditional computing resources.

Graphics Processing Units (GPUs) have been evolved into accelerators for high-throughput and data-parallel jobs [40-42]. Originally designed for gaming purposes, accelerator-based GPU computing has been adopted for many compute-intensive scientific applications due to its highly parallel architecture, such as accelerated feature finding in proteomics datasets generated by MS [43], simulation of cardiac tissue in real-time [44], accelerated simulation of ECG [45], GPU-accelerated MRI reconstruction algorithms [46], real-time detection of biological targets using GPUs [47], parallel mining of neuronal spike streams coming out of MEAs [48]. GPU-based computation provides not only faster but economically cheaper solutions to the traditional setups. Thus, it is natural that commercialization of accelerators is enabling their use in analyzing large-scale data within a fraction of budget of comparable traditional machines. Recent efforts have also shown significant performance improvements by integrating advanced I/O techniques with accelerator-based GPU processing for high-performance real-time computing [49, 50].

An end-to-end system that encompasses bio, nano and info domains is required to acquire, process, and analyze the data towards the detection of biological targets, which can ultimately help in

disease diagnosis. Such work will lay the foundation for automated disease diagnosis as compared to the state-of-the-art manual processing associated with bio-nano devices.

1.3. Objectives and Approaches

In order to automate the process of manually analyzing the data to detect biological targets, there are couple of challenges tied to it: (i) selection of threshold to efficiently detect patterns and discard the noise, (iii) online processing of the said data for realizing instantaneous feedback, and (iii) useful feature computation, and classification of biological targets in the raw data.

To decide the right threshold, we will use different threshold techniques including static and dynamic techniques and compare the results to find the optimal out of these techniques, this will address challenge (i). However, the static threshold techniques are simpler and faster, but easily results in false alarms due to noise and baseline wanders. In contrast, the dynamic techniques are relatively complex, but result in efficient detection of pattern even in the presence of noise and baseline artifacts. To enable fast and efficient processing of the data from a large number of bio-nano sensors, that is, addressing (ii), we employ parallel implementation of the detection techniques on GPUs coupled with advanced I/O techniques. The parallel implementation of the techniques makes it faster to process the data as compared to single threaded implementation. In addition to that, the advanced I/O techniques, such as double buffering coupled with parallel implementation overlap I/O with the computation, and further improve the performance. Such implementation results in a faster detection platform for the detection and analysis of biological targets, and matches the actual throughput of couple of bio-sensors collecting data from many biological targets, simultaneously. Thus, faster computational setup provides room for future bio-nano sensing setup in terms of catch up, which we foresee as a library of sensors where each can support translocation of biomolecules and therefore, results in many times faster sampling speeds than a single desktop computational setup. To address (iii) we adapt pattern classification techniques in order to classify different biological targets that are detected in the raw data. However, the classification of targets

into separate classes requires significant features that will help in accurate classification of the targets and further help in disease diagnosis.

In a typical clinical experiment, few milliliters of blood sample translocation results in few GBs of raw data. Thus, the collected raw data from a hundred patients visiting each day can easily result into TBs of data. Such data is beyond the limited resources of a typical machine, and can easily overflow the limited resources of a typical desktop machine. Furthermore, the limited memory cannot accommodate the data that is beyond a few GBs. To mitigate these bottlenecks, we adapt a streaming approach and advanced I/O techniques, such as, double buffering to make the overall process efficient. The streaming approach streams data into chunks from the storage to the main memory, process it, discard the original data, and finally, write back only the results to the storage. Furthermore, the results, that is, the detected biological targets in the blood samples, roughly constitute a thousandth fraction of the patient blood sample data. Discarding the original raw data, and processing, and storing the important patterns not only results in a faster software platform but also result in efficient utilization of the limited resources of a machine.

In a nutshell, the goal of my research is towards the design, and development of software techniques that can adapt to the changing data characteristics of bio-nano devices, and thus, automate the detection and classification of biological targets in the raw data. This research will lay the foundation for automated disease diagnosis at early stages using bio-nano devices.

Chapter 2

Literature Review

In this chapter, we present background of enabling technologies, and the work related to the research proposed in this dissertation. The related work discusses pattern-detection algorithms in order to automate the process of detecting interesting patterns across different domains and the challenges tied to them. Furthermore, we showcase few examples from biomedical and biophysical applications that suffer from the problem of data deluge, and the use of pattern-detection techniques, and graphics processing units (GPUs) towards automation and accelerated processing.

2.1. Pattern-detection Algorithms

Pattern-detection is a critical area of research in biomedical and biophysical applications, and a significant amount of work is focused on pattern-based work applied towards detecting interesting patterns in such applications. Different pattern-detection algorithms are applied across different domains; however, there are still challenges that are common to many pattern-detection algorithms.

The problem of detecting patterns exists across diverse applications including mass spectroscopy, signal processing, image processing and bioinformatics [11, 12, 15, 16]. To address this problem, many pattern-detection algorithms have been developed, unfortunately, each of them work only in their target domain [11, 12, 15-17, 19, 20, 51, 52]. Therefore, such coupling prevents the interoperability and reusability of the algorithms across different domains. Efforts have been done to formalize a

mathematical model [51] for peak detection, however, it still suffers from large number of false alarms unless tailored and tuned for a particular domain.

General approach to detect patterns consists of three steps — (i) to get rid of noise by smoothing, (ii) the baseline correction, and (iii) peak detection in the smoothed data as captured by Yang et. al [11]. The goal of the smoothing is to highlight the peaks and suppress the noise, and weak peaks. The baseline correction get rids of the baseline shifts, and brings the time-series data to a stable state. This improves the precision of peak detection process by choosing a threshold on a less noisy data with removed baseline artifacts. The pattern-detection algorithms include all, or a subset of these steps in order to accomplish the task of detecting patterns.

The key challenge is the noise in the data. Furthermore, the nature of peaks occurring at different scales and amplitudes further makes the task of peak detection challenging, and easily runs into false positives in the detected peaks. Furthermore, peaks can have characteristics shapes, such as the observed peaks in mass spectroscopy. To address these challenges, wavelets have been used to detect peaks in the spectroscopy data [10, 53]. Pattern-detection algorithm based on Continuous wavelet transform (CWT) is also used for peak detection [12]. Aggregated monitoring is done based on window-based models, such as landmark windows, sliding windows, and damped windows [54-57]. However, the sliding window is widely adopted in the data stream monitoring. Therefore, it has also been generalized to the elastic window model. In order to find abnormal aggregates in the data streams, wavelet-based burst detection is proposed by [58]. The haar wavelet coefficients are stored in special data structure i.e. shifted wavelet tree (SWT), where each node corresponds to a specific window size and each level in the tree corresponds to a specific resolution. The appropriate size of window and resolution for a specific burst is selected by scanning the SWT. Such bursts exist in Gama Ray data streams and vary widely in their duration and strength. However, selecting thresholds for different window sizes is still a future work.

Inspired from Newtonian mechanics, a momentum-based algorithm is developed that computes changes in the momentum as the ball traverses the time-series [59]. Such peak detection algorithm can

find both peaks and troughs of a signal. The algorithm computes momentum at various points and determines the peaks and troughs without using smoothing and thresholding. In case of highly noisy signals, the data will require smoothing to remove the high frequency noise. Nonetheless, adding friction and initial momentum as tunable parameter, so that the user can fine-tune the algorithm for a specific data set – can help in eliminating the smoothing step.

To mitigate the impact of noise, the signal to noise ratio (SNR) needs to be above certain threshold [52, 60]. However, we are still left over with the key challenge how to set select the threshold in order to minimize the false alarms, and thus, have an acceptable trade-off for precision and recall that varies from domain to domain. One effort is to decide the threshold based on the noise-level in the data that stems from the mean and standard deviation in the data, such as h = (max + absolute_mean) / 2 + K * absolute_deviation, where max is the maximum value in the time-series, absolute_mean is the average of absolute values in the time-series, absolute_deviation is the mean absolute deviation, and K is the influence factor of deviation. In other words, K is user-specific constant. It is clear from the above equation that greater standard deviation results in higher threshold and vice versa.

Another work includes peak detection in gene expression from microarray time-series data, followed by support vector machine (SVM) based classification into its functional groups [17]. Multiple methods based on score assignment were used to detect peaks, such as score is assigned based on the fraction of area under the candidate peak, or based the rate of the change calculated at each point i.e. the derivative. Peaks that were selected from multiple methods were identified as true peaks, which were fed to SVM for classification. Moving-based algorithm is designed to detect peaks such that values larger than x times the standard deviation of the time-series are considered as peaks. The extent of smoothing is empirically decided, such as 24 points for daily data.

In addition to filter-based techniques and spike-shape interpolation, several techniques have been established to analyze neuron action potentials including feature analysis, principal component analysis, cluster analysis, Bayesian clustering, clustering in higher dimensions, and template matching [33]. In all

cases, the value of the threshold is crucial to determine the signal (above/below the threshold), and noise (above/below the threshold), and vice versa. Different values of the threshold result in different signal to noise ratios that could ultimately lead into different trade-offs between precision and false alarms.

In addition to the challenges mentioned above, the pattern-detection algorithms depend on a single, or multiple threshold values. This further decides and, evaluates the effectiveness of the devised algorithms. Furthermore, the extent of smoothing depends on the level of noise in the data. Too much smoothing can not only remove noise, but also remove small peaks which might have useful meaning in the domain. The final criterion that decides whether the data point is a peak or not, always results from the comparison to a threshold. To summarize, the data generated from the above-mentioned applications is noisy and riddled with baseline artifacts – it is crucial for each application to define a criterion that can determine the threshold dynamically, instead of a one-shot static threshold, which due to slight variations in noise level, or baseline can easily result in false alarms. Attempts have been done to compute threshold dynamically [18]. Unfortunately, these algorithms work under some constraints, such as the absence of outliers in the data [18], reliance on a minimal threshold [59], additional parameters, such as the length of window [34], and algorithmic specific parameters including minimum momentum [59], where a constant value will not suffice for every situation. These optimizations further add to the complexity of already dynamic environment, and unpredictable data characteristics mainly due to baseline artifacts and noise.

2.2. Graphics Processing Units (GPUs)

The recent trend has shown a significant increase in the use of *Graphics Processing Units* (GPUs) as state-of-the-art accelerators, for compute- and data-intensive computing; such as feature detection in large amounts of protein data [43], fast mining of huge spike trains generated from Multi-Electrode Array [48], and other useful GPU applications [41, 46, 48, 61-63] within a fraction of a budget of the traditional setups. Commoditization has become commonplace for asymmetric processors and heterogeneous architectures with offload processing engines [42]. Recent efforts have also shown significant

performance improvement by integrating advanced I/O techniques with accelerator-based GPU processing [49] [50].

2.2.1. CUDA

The programmers can harnesses the underlying massive parallel architecture of the GPUs through parallel programming tools. A forerunner manufacturer of GPUs, NVIDIA Corporation, has developed Compute Unified Device Architecture (CUDA) [64]. CUDA is a parallel computing architecture used on all NVIDIA products, such as Telsa [65] and Fermi [66]. The access to the parallel computational elements in CUDA-GPUs is provided through various application programming interfaces (API). The software developers can use variety of standard programming languages (like C, C++, C#) to access CUDA via APIs. This provides abstractions to the programmers to implement parallelism at different levels of granularity. The programmers can express coarse-grained, fine-grained or task-parallelized abstractions using high-level languages. A marvelous introduction to data-parallel GPU programming in CUDA has been provided by Nickolls et. al [67].

The CUDA programming model follows Single Program Multiple Data (SPMD) style of parallel programming. SPMD allows programmers to write thread level parallel, or data parallel code for independent or coordinated threads, respectively. SPMD further enables specifying the execution behavior of an individual thread. Thus, the kernel functions are launched on GPU to execute compute-intensive and data-parallel code in parallel. The CUDA kernel is composed of light-weight threads that are grouped into blocks and the blocks are further grouped into the kernel grid as shown in Fig 4. The execution configuration for kernel-launch typically consists of 64 to 512 threads per block and a number of blocks in the kernel grid. However, in newer Fermi-based GPUs maximum number of threads per block reaches up to 1024. The thread block is further composed of batches of 32 threads called warps, such that number of warps per block is number of threads per block divided by 32. Shared memory within a block is visible to all the threads in the block and hence all the threads are synchronized. However, the

threads across different blocks do not have access to *shared memory* and are not synchronized. As for global memory, all the threads within a kernel launch can access it. Additionally, each thread has its own private local memory. To cope with branch divergence issues where half of the threads are processing one piece of code and half working on another within a thread block, synchronization per thread block is needed. Such synchronization makes sure that all the threads in the block are finished and have reached a destined barrier. To this end, the programmer specifies the number of threads per block and the number of blocks per grid in the CUDA API for kernel launch. Furthermore, a programmer can alter the execution configuration of a kernel launch by changing the block dimensions. This can be done by indexing threads per block in 1- (x), 2- (x, y) or 3-dimensions (x, y, z) subsequently forming 1-, 2- and 3-dimesional (D) thread blocks respectively. The indexing scheme of threads for 1-D block stays the same. However, for higher dimensions it is $x+yD_x$ and $x+yD_x+zD_y$ for 2-dimension (D_x, D_y) and 3-dimension (D_x, D_y, D_z) block sizes, respectively. Here D_x, D_y, and D_z pertain to the dimensions along x-axis, y-axis, and z-axis respectively. Irrespective of the different dimensionality of threads in blocks, the total number of threads should not exceed the maximum limit imposed by its hardware architecture, as already mentioned. Thus, different execution configurations can be launched to map different target applications on a GPU for improved performance. Single kernel can be launched instantaneously to the GPU in older architectures with compute capability less than 2.0. Multiple kernels are serialized and executed one by one on the GPU. Also, the execution configuration can only be launched by the programmer from CPU to GPU, and once launched, the configuration never changes for that particular kernel launch, supporting *static parallelism*. However, multiple kernels (up to 4) can be launched to execute concurrently with compute capability 2.0 and higher, such as *Fermi* [66] and *Kepler* [68] as shown in Fig. 5. Additionally, the execution configuration can be changed on fly without the intervention of the CPU in Kepler-based setup and therefore, facilitating *dynamic parallelism*.

2.2.2. OpenCL

OpenCL [69] was introduced as an open standard language for programming GPUs first broadly supported by all GPUs and similar devices. In addition to portability across different platforms, different architectures still vary in their memory requirements, and related optimizations to achieve the peak performance. Similar to CUDA, it consists of C/C++ part that run on CPU and the C part that run on GPUs. The CPU code is used to control the overall functioning of the parallel code that runs on GPU including memory allocation, launching the kernel, and finally, gathering the results. The GPU kernel follows Single-Program Multiple-Data (SPMD) style of coding. Similar to CUDA each kernel consists of 2D and 3D grids. The difference only lies in the terminology, such as the thread-blocks referred in CUDA are known as work-groups in OpenCL. The workers (threads in CUDA) can synchronize with one another, but not across the work-groups. From CPU perspective, the basic and smallest unit of execution is the kernel that could be launched on the GPU. CUDA vis a vis OpenCL isolates the main memory from the side effects of updates to local variables by happening them only in the device memory. An application program can manage the attached accelerator through language extensions, such as "Jacket" for Matlab[1], or PyCUDA[2] for Python, which contain libraries including BLAS and FFT routines[70, 71].

2.3. Biophysical and Biomedical Applications

It is common problem that most of the biophysical and biomedical applications suffer from the problem of processing raw data generated by sensing technologies. Handling raw data and finding interesting patterns in it towards useful analysis is too subjective and inefficient. Efforts have been done in order to automate such processing towards real-time processing and analysis. The following sections show cases

[1] http://www.accelereyes.com
[2] http://pypi.python.org/pypi/pycuda

such biophysical and biomedical applications and discusses the performance improvement achieved through automation using graphics processing units (GPUs), and the future challenges.

2.3.1. Automation of Mass Spectrometry Analysis

Mass spectrometry (MS) is one of the techniques used for the expression analysis of proteins and has several applications in medical diagnostics, biomedical engineering and therapy. This technique is used to determine the mass, composition and structure of the target molecules especially peptides (digested proteins) for decision-making. The peptides are ionized and accelerated within mass spectrometer to get the fingerprint (intensity vs. mass).

The 3D contour maps can also be generated, with x-axis showing mass-to-charge ratio, y-axis representing the intensity and z-axis for additional parameters. However, one scan can easily overflow over several gigabytes. Furthermore, the manual analysis and interpretation of useful features in the measured data is significantly larger job than the MS experiment itself due to the inherent noise and baseline artifacts. To automate the process towards faster processing, adaptive wavelet transform has been implemented on GPU for feature detection [43]. The ideal scenario is to have high precision and performance. However, there is always a trade-off between performance and precision. Data can be pre-processed for noise removal using simple thresholding to detect the features in protein data. Additional preprocessing could be done to get rid of noise and get better quality results e.g., morphological filters or smoothing techniques. Moreover, additional post-processing includes clustering and model fitting techniques, but makes the process computationally expensive. The wavelet transform is robust to the noise by design and efficiently removes inherent chemical noise in the data. The transform slides over the preprocessed data and convolves kernel function to it. This makes the algorithm embarrassingly parallel and maps well to the massive parallel architecture of GPU. The convolution is performed in parallel by as many threads as data points in CUDA. The algorithm divides the input map of scan points into chunks of 512 points (maximum number of threads per block in older architectures), and assigns each point to the

individual thread. Each thread performs a convolution operation on the corresponding point to its neighboring points. Actually more points are loaded into shared memory to facilitate convolution of neighboring intensities to the assigned point per thread. Unfortunately, the number of neighboring points to be loaded for a particular convolution of a data point can't be determined in advance because of the irregular spacing in MS data. Therefore, maximum amount of memory is reserved to load all the associated neighboring points based on the maximum mass. However, this incurs memory pressure due to the limited size of shared memory.

Two GPU-based algorithms have been implemented using NVIDIA Tesla C870, hosting 128 cores, each operated at 1.35 GHz with a global memory of 1.5 GB and shared memory of only 16 KB [43]. The counter CPU implementation used 8 cores with each operated at 2.3 GHz and 16 GB main memory. To further accelerate the process, another implementation used on-chip shared memory. The GPU-based algorithm even with no approximations was not only faster, but also resulted in better quality results. This led to 200X speedup on real-world protein data as compared to its counter CPU-based implementation. Additionally, implementation on slightly older architecture of NVIDIA Quadro NVS 290 resulted in a speedup of 10X. Given these results, larger scale proteomics, where larger number of proteins needed to be detected, would necessitate the use of advanced Fermi and Kepler-based GPUs. Such GPUs have the potential to accelerate the processing of such highly parallel algorithms e.g., convolution to happen in real-time due to their massively parallel architecture due to increased number of processing cores, number of threads per block and shared memory.

2.3.2. Accelerated Medical Imaging

Many neurological disorder and heart problems can be detected, in vivo through MRI. Such imaging can get images from different softer tissues of the body with better contrast than X-rays or Computed Tomography (CT). The data accumulated from imaging can be reconstructed in any plane with minimum loss in the image quality. The challenge in automating the MRI process is the high frequency noise in the

data, imaging artifacts and extended data acquisition times. The scanning trajectory profoundly affects the quality of reconstructed images. The trajectory can be cartesian or non-cartesian, however, the latter is becoming more common in MRI due to its robustness towards noise. The non-cartesian trajectory samples are first interpolated onto a uniform cartesian grid, and then reconstructed via fast Fourier transform (FFT) [72]. Such computation is easy but does not incorporate anatomical information. However, incorporating this information helps in reducing the noise by achieving higher signal-to-noise ratios per scan in optimal image reconstruction, without sacrificing the quality (resolution) of the image features. [73-76]. The advanced reconstruction algorithms designed for large scale problems can be real-time solutions in clinical setup if accelerated on GPUs. For instance, advanced reconstruction of non-uniform scan data takes less than 2 minutes on GPU as compared to its CPU-based gridded reconstruction which takes 23 minutes as shown by Stone, et al. [77].

The advanced reconstructions algorithms are not only faster but are also more precise e.g., 12% error was reported when compared to its counter CPU-based implementation which had 42% error. The experimental system consisted of NVIDIA's Quadro FX 5600 GPU operating at 1.35 GHz coupled with dual-core Opteron 2216 CPU. The GPU-based implementations for reconstruction algorithms include naive implementation, and well-tuned optimized implementations. Note that the optimization parameters vary across different applications and mainly depends on the inherit parallelism in the data. However, in this case such tuning significantly reduced the off-chip memory bandwidth by reducing the number of memory accesses to the global memory and resulted in improved performance i.e. runtime reduced to 59 seconds. Additionally, running the tuned implementation on multi-GPU setup resulted in a total runtime of 18 seconds. Note that single-precision floating-point computation along with approximations of trigonometric operations though resulted in acceleration of the method and less error towards the real image; however, these could also degrade the quality of the final results. Improving the quality of the acquired images in addition to the acceleration implementation can increase the scanner throughput and

hence reduce the patient discomfort. Furthermore, speed-up of 2X to 9X for the same reconstruction algorithms has also been achieved [72, 78, 79].

Another important issue in MRI is the performance of automated image registration algorithms. The purpose of automated image registration is to find mapping between the source image and the reference image using transformations and similarity measures followed by optimization to find the best transformations. The problem with these algorithms of image registration is that these involve significant iterations and are quite slower on a typical workstation. This delays the post-processing of medical images and hence increases the response time in medical diagnostics. Acceleration of up to 14X for image registration calculations of MRI on GPUs has been achieved [80]. However, acceleration based on latest GPUs is yet to be seen.

2.3.3. Real-Time Processing of Raw data in Electrocardiography

ECG is a well-known technique to record the electrical activity of the heart. ECG is more pervasive than MRI and CT with no radiations and has little discomfort to the patient. This provides 3D and even 4D views of the heart. Each echo coming out of human heart needs to be connected to the ECG in order to record the target cardiac process of the human heart. The visualization of higher dimensions of images is called volume rendering. The problem with this application is that the volume of the raw data acquired is huge, and rendering the higher dimension images in real-time is computationally-expensive and time-consuming. It is hard to keep pace with rapid heart-beat rate, while acquiring and rendering huge volumes of data per cardiac cycle. However, GPUs can be exploited for rendering higher dimensional images of the acquired ultrasound data and can almost reach the audible heart rate [79]. GPUs have been used to achieve 13.03X faster computations than its counter sequential code [45]. NVIDIA GeForce 8800 GT engine hosting 112 cores, operated at 1.5 GHz with a device memory of 512 MB, was used for GPU-based implementation, compared to the CPU operated at 2.4 GHz with 3 GB of main memory. The need of more data for GPU computation actually hampers further acceleration. However, advanced GPU-based

setup can result in improved performance with massive number of cores, larger memory and higher memory bandwidth between the CPU and GPU.

Ray-casting is also one of the techniques used for volume-rendering. This technique transforms 2D data into 3D projection by tracing out rays from the view point into the viewing volume. In simple words, ray-casting is a technique to render high quality images of the solid objects. A framework for ultrasound datasets that uses GPUs towards ray-casting algorithms for volume rendering has also been developed [81].

2.3.4. GPU-Accelerated Analysis of Neuron Action Potentials

Electroencephalography (EEG), functional magnetic resonance imaging (fMRI) and multi-electrode arrays (MEAs) are all used for the analysis of neuronal activity of human brain. These approaches do not allow recording of a single neuron activity. Single-unit recording measures electrophysiological activities up to the resolution of a single neuron, measuring intracellular and extracellular events of human brain, called the action potentials. Cao et. al has used data mining algorithms towards the event streams (neuron action potentials) coming out of MEAs to study the functionality of human brain [48]. The problem in analyzing these interesting patterns is that the accumulated event streams from MEA have tremendously increased in size where a typical 64-channel MEA can easily measure millions of spikes within a couple of minutes. However, the actual experiments can run up to months which generate trillions of neuronal action potential spike data. To this end, such data deluge entails huge data storage and high-end computation power. This motivated the design of data mining algorithms to analyze the MEA recorded event streams in real-time. Furthermore, the GPU-based solution was not only a real-time approach but also an alternate economic solution to supercomputers by providing their implementation on a typical desktop computer housing GPUs. The task of finding frequency of non-overlapped occurrences of patterns called frequent episodes in the event streams was parallelized. Different computation-to-core mapping resulted in different speedups: One thread per occurrence was used to fulfill the mining of fewer

numbers of episodes, while the two-pass elimination was done to count larger number of episodes. The problem with the former approach was the under-utilization of GPU due to the data dependencies and branch divergence. The later approach assigned only one episode per thread. However, greater number of episodes fully utilized the GPU. Unfortunately, only 32 threads could be scheduled on the GPU's multiprocessor which hosted just 16 KB of registers and shared memory. Given such a small number of threads that could be scheduled at a time on GPU, little performance enhancement was achieved. However, reducing the complexity of the algorithm without sacrificing the correctness helped in enhancing the performance. The relationship between frequency of episodes and its counting time followed a linear trend, and the running time of the algorithm became dominant by the sorting time with the increase in episode frequency.

The two-pass elimination algorithm showed speedup of 1.2X – 2.8X over the hybrid approach for different datasets at different support thresholds. In fact, branch divergence and local memory were responsible for the improvement in two-pass elimination, which stemmed from tracking the episodes. More specifically, the former algorithm eliminated 99.9% of unsupported episodes of size four resulting in speedup of 2.53X. The two-pass elimination algorithm used less number of registers and no local memory, as compared to the hybrid algorithms. Moreover, the number of memory accesses was fewer in two-pass elimination method, when large number of episodes was used. The algorithms were evaluated on NVIDIA GTX280 GPU with 240 cores clocked at 1.3 GHz with 1 GB device memory, and an i7-based CPU with a clock frequency of 2.33 GHz and 4 GB of main memory.

2.3.5. Real-Time Analysis in Carbon Nanotubes

Another application area that necessitates the use of real-time analysis is the processing of large amount of neuro-sensor data coming out of assistive technologies [26]. The duty cycle of neuron action potential varies from a fraction of a millisecond to few milliseconds [33], and therefore, needs sampling rate of few kHz. However, measuring discrete voltage signals with say thousands of electrodes, each operated at 1

kHz of duty cycle requires at least a million data points to be recorded faithfully per second. Consider an approximate resolution of 8-bits leads towards 256 distinct voltage levels per electrode. The system has to be capable of an actual measurement rate of at least 1 MB/sec. Given such a large number of electrodes (10K to 50K per μm^2), necessary for effective measurements, and the need to adapt faster rates of action potentials easily overwhelms few GB/s due to the aggregated data rates. Unfortunately, the traditional approaches with very few electrodes rely on dumping raw data to storage devices for later processing incurs long turn-around delays between the actual collection of data and to generate useful electrical stimulants. Such unacceptable delays in assistive technologies can be mitigated with high-performance GPUs for recording action potentials in real time.

2.3.6. GPU-Based Detection of Biological Targets Using Nanopores and Micropores

Nanopore biosensors have been used to study various biophysical properties of biologial molecules [23, 82-84]. In nanopore studies, translocation of biomolecules are monitored using a dual compartment setup, separated by a membrane with steady-state ionic current flowing in between. Biological molecules inherently carry charges and are thus translocated through the nanopore under the electrophoretic force of the applied bias. During translocation, the biomolecule physically blocks the pore, resulting into significant current blockade, called a pulse. Different hypotheses have been suggested based on pulse features, such as pulse-shape and width, where shape relates to the orientation of the target, while the width of the pulse relates to its length [21, 83, 85]. Pulses with significant distinctive characteristics have been observed in the case of DNA molecules that are different by as few as one to three bases [23].

A micropore device works on the same principle except the size of the pore is larger and bigger to translocate biological entities, such as living cells. Diseased cells like tumor cells are known to be mechaicanlly different than normal cells [39, 86-88]. Tumor cells have been found to be more elastic than other cell types [22, 89-91], and significantly different current blockade pulses were recorded in the case of tumor cells [39].

The detection system recordes the data at high sampling frequency (few MHz) resulting in millions of data samples recorded per second, in order to record the biological interactoins faithfully. Additionally, the raw data genrated by these sensors is riddled with inherit noise and baseline artifacts. Furthermore, the manual analysis on the recorded data is very tedious, time-consuming, and error-prone. This will become more challenging when the future nanopore design will include an array of nanopores, such as 2x2 nanopore matrix. Such matrix when used for DNA translocation running for an hour, with a sampling rate of 400,000 samples/second, and using 16 bits to represent each sample, will end up in approximately 11 GB of data. To this end, typical workstations will not suffice, and we will need techniques, such as streaming and buffering to alleviated the I/O bottleneck. However, parallel algorithms will be required to mitigate the computational bottleneck for processing the raw DNA data. GPUs have been adopted for real-time processing and analysis of the bio-targets from micropore-based detection system [92]. However, the algorithm doesn't show significant speedup on GPU due to the branch divergence in case of moving-average filtering algorithm. Futhermore, the algorithm starts by first loading a chunk of data equal to the window size of the algorithm resulting in significant memory accesses to global memroy. This incurs memory latency, and hence hampers the performance gain on GPU. The use of advanced parallel algorithms that better tailor to the GPU architecture coupled with CUDA-based optimizations will provide better speedups.

Chapter 3

Detection and Classification of Biological Targets in Bio-Nano Sensing

To address the problem of manual detection in the data deluge, a high-throughput computer-based automated system is required, which can detect pulses efficiently and accurately in real time. Our preliminary work proposes a novel solution by coupling advanced I/O techniques with GPU-based parallel algorithms for analyzing the large datasets produced from solid-state nanopores and detect biological targets in real time [92]. This approach has shown significant improvement in real-time pulse detection and can be used as an integral part of the next-generation production systems.

Specifically, this work makes the following contributions:

- Design and implementation of a static and two variations of dynamic pulse-detection algorithms (*baseline-tracker* and *moving average*). Each algorithm analyzes the raw input signal and detects the biological target by identifying the pulses in the input signal.
- Use of advanced I/O techniques, such as double buffering and asynchronous I/O for overlapping I/O with the computation to minimize the I/O bottlenecks. The system is fully pipelined such that it streams raw data into the memory and after pre-processing the data, it is processed for pulse detection at GPU, and the final results are reported to the user in a human-readable format for visualization.

- Additional features are computed in order to minimize the overlap among the observed biological targets.
- Machine learning techniques towards Cancer cells classification with and without training data.

- Two-step detection technique: Moving-average based smoothing, followed by the threshold that is based on the statistics of the data in order to adapt to the varying characteristics of the data and thus, effectively detect biological targets in them.

3.1. GPU-Based Detection of Biological-Targets in Raw Solid-State Pores Data

3.1.1. System Overview

Details of the implemented algorithms and the parallelization of the developed algorithms on CUDA-enabled GPU are described here.

A high-level overview of our designed system is captured in Fig 2. The system consists of different software modules such that each module implements a distinct functionality. The first module, the Data Pre-fetcher streams raw data into the host memory using double buffering. Data-formatter receives data from Data Pre-fetcher and converts the streamed data to the integer format for computational accelerator (GPU), and passes it over to the GPU-Manager. The GPU-Manager off-loads the buffered data to the device memory using asynchronous I/O, launches the GPU kernel and finally asynchronously copies the results back to the host memory.

The main core (host device) accumulates, and merges the results, and produces the consolidated result. The Result-Analyzer collects the data from the device memory, converts it to the required format i.e., width and amplitude (peak) of the pulse, and finally delivers it to the user in the form of scatter plots.

3.1.2. Algorithms for Pulse Detection

The main goal of the system is to efficiently detect pulses from the given dataset in real time. Different hypotheses have been suggested based on the *pulse-shape* and *width*, where width of the pulse relates to the molecule length, while the shape of the pulse, i.e., its duration and peak relate to its orientation [93, 94]. *A pulse is detected when the current signal (baseline) falls below a certain threshold of the baseline current, and reverts back to the original stable level after a certain time (pulse duration).*

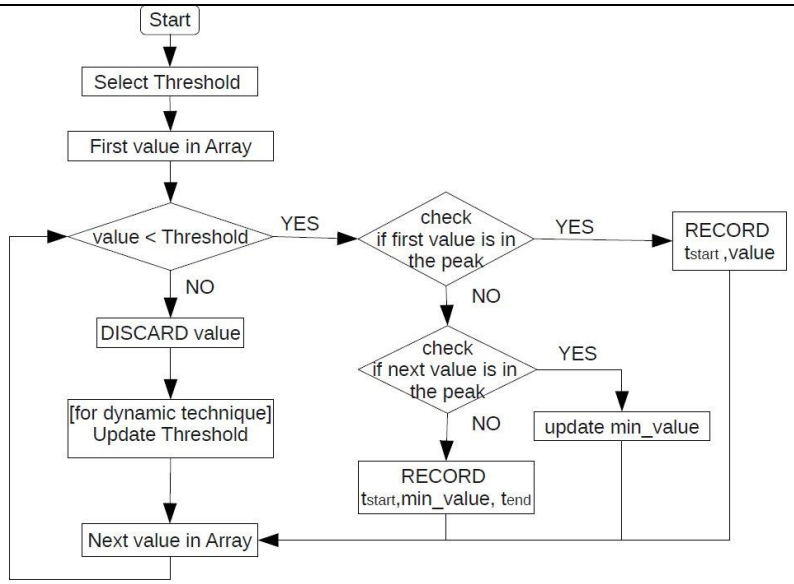


Fig 1. General flow chart of the pulse detection algorithms

The system detects the pulses against a pre-selected threshold based on the given criterion of acceptable range of the pulse width. Three pulse-detection algorithms are designed that can effectively detect the pulses from the given data. Fig 1 shows the general flowcharts of pulse detection algorithms. The algorithms differ from each other in the selection criterion of the threshold depending on whether it is static or dynamic. Dynamic algorithms further differ in the adapting nature of the threshold, during the process of detection. The updating step is only for the dynamic techniques.

A. Fixed Threshold Technique

The fixed threshold technique processes signal values from the given dataset, and compares them against a given fixed threshold. The threshold is selected at the beginning of the detection process with some difference *x* (in nanoamperes) below the baseline. This threshold remains fixed throughout the detection process. If the *x* is selected too high, then pulses only with amplitude larger than *x* are detected (results in false negatives). Selecting *x* too small is also not favorable as this results in false positives (detects pulses with much smaller amplitude which is part of the background noise) [95]. All the data points are tested against this static threshold and are recorded only if it is less than it. The detected pulse can consist of a single data point, or it can be a group of data points. The pulses that have very small duration (e.g., less than three sampling periods) or those with very large duration (e.g., more than a hundred of sampling periods) are considered noise. When a value less than the threshold observed, the pulse-detection process starts. The start time and corresponding current value are recorded. The minimum signal value is tracked during the pulse detection process. When a value greater than threshold is encountered, the individual pulse-detection completes and the start time, end time and the minimum of the pulse is recorded. The width of the pulse is calculated from the end time and start time of the detected pulse. The pulses that fall in the acceptable range are recorded only. For our problem, the *Min* and *Max* of the acceptable range for pulse width is from 150 microseconds to 1000 microseconds when data is sampled at 75 microseconds; while for data sampled at 2.2 microseconds, the acceptable range varies from 5 microseconds to 1000 microseconds. The *Min* is decided based upon the fact that there should at least be two samples contributing towards the pulse width. The *Max* value is selected based on the type of biomolecule being processed. For our problem, we know from the data characteristics and previous knowledge that the maximum pulse width of any significant translocation should be less than 1000 microseconds. Once the sampling rate is optimized and the pulse width range is selected, the criterion to reject/accept any pulse stays the same. Furthermore, from the domain knowledge we found that the threshold of 1200 nanoamperes below baseline is optimum for our dataset. The reason is that the amplitude of the noisy

pulses is normally smaller than 1000 nanoamperes. In this way pulses are detected precisely and results of the automated approach are verified to be in very good agreement with human-based decisions about the detected pulses.

The fixed threshold technique is the simplest approach that is designed for pulse detection. This approach is faster but error-prone when the baseline is not stable. It may ignore the valid pulses (false negatives), or may detect noisy pulses (false positives) due to the variation in the baseline. To address this problem, adaptive (dynamic) threshold technique is devised. Instead of using a fixed threshold, an adaptive threshold is used that is updated at each signal value, except while analyzing an individual pulse.

B. Dynamic Threshold (Baseline-tracker technique)

The baseline-tracker technique use an adaptive threshold, which keeps track of the baseline with difference x (nanoamperes) below it and detects pulses accordingly. This starts with a seed value that is less than the initial current value with a difference x. How much x the threshold should be kept below the baseline is decided manually by observing the noise value. Once x is optimized, there is no need to modify it further. This threshold is tracked in accordance with the baseline as temporal data is processed to detect pulses in it. The pulse-detection process starts when the signal value drops abruptly from the baseline and goes down the threshold value. Note that the threshold is updated at each value of the baseline except for the duration of pulse detection process.

To elaborate the detection process we use the following example. For instance, we have a sequence of signal values: a, b, c, d, e, f, g, with T as the threshold; let's suppose $(c, d, e, f) < T$, and $(a, b, g) > T$. For signal value a: since $a > T$, so T is updated as, $T = a - x$; where x is the difference by which T is kept below baseline for each signal value greater than threshold. We repeat this process for the next signal value, b which is also greater than T. For subsequent signal values; $c, d, e,$ *and* f, that are smaller than T, we record the t_{start}, *min* and t_{end}. After calculating the pulse width from t_{start} and t_{end} we record it only if its width fall in the acceptable range, as described in Fixed threshold technique. The analysis of a pulse completes, based upon two conditions. First, when the baseline go above the threshold and second, when

the baseline stays below the threshold for a time greater than the acceptable range of pulse width. In the latter case the pulse under analysis is discarded. In this way, all the real pulses whose width falls within the acceptable range are recorded.

C. Dynamic Threshold (Moving Average Technique)

Moving average technique is another dynamic threshold technique that is adapted for the pulse detection process. In statistical terms, this technique is also known as running average. It maintains a running average calculated over a set of numbers, also called window *w*. The threshold is calculated based on that running average with some tolerance as explained below.

For example for a window size $w=5$, the running average is given as:

$$average_5 = (y_{t-4}+y_{t-3}+y_{t-2}+y_{t-1}+y_t)/5 \quad (1)$$

For subsequent temporal data points, t = 6 and t = 7, Eq (1) becomes:

$$average_6 = (y_2+y_3+y_4+y_5+y_6)/5 \quad (2)$$

$$average_7 = (y_3+y_4+y_5+y_6+y_7)/5 \quad (3)$$

To reduce the number of computations, we keep a running sum of the *w* most recent temporal data points. At every iteration new value occurring at *t* is added and the oldest value is subtracted from the sum.

$$sum_6 = y_6+sum_5-y_1 \quad (4)$$

$$sum_7 = y_7+sum_6-y_2 \quad (5)$$

Then the average simply becomes: $average_t = sum_t/5$.

We then compute the threshold *T* from the running average with some tolerance *x*% i.e.

$$T = average_t - average_t \times tolerance. \quad (6)$$

The moving average algorithm detects pulses by comparing every signal value against the computed threshold. This technique differs from the Baseline-tracker technique in the computation of *T*. The individual pulse analysis (finding start time, end time and minimum among the detected points) and the acceptable range of pulse width are same for both methods.

We have observed the pulse-detection with different window sizes, and tolerance. When *w* is too small (e.g., 10 items), the algorithm runs faster but results in false positives. When *w* is too large e.g., 1000, the algorithm runs slower but results in better detection of pulses. For our dataset, the best match between the speed and quality of pulses is achieved with window size of 200 and tolerance of 2.8%. As compared to other techniques moving average output matches manual detection output with a difference of 8.1% at the cost of increased computational complexity.

3.1.3. Detailed Architecture

The major software components of the developed system are as follows:

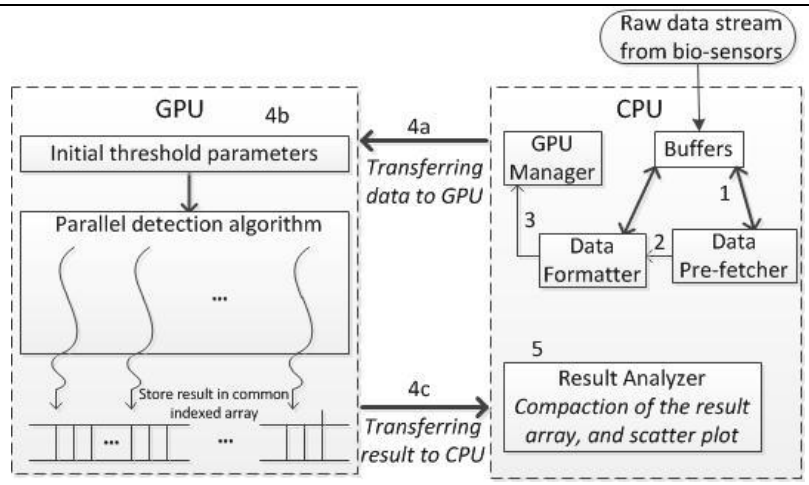


Fig 2. System architecture with its major components including Data pre-fetcher, Data formatter, Pulse detector, and Result analyzer.

A. Data Prefetcher

The Data Prefetcher uses double buffering technique to overlap I/O with the computation. In double buffering technique, two separate buffers are used as shown in Fig 2. As the first buffer is being processed by the processing thread, the reader thread reads the input data from the storage device in the second buffer. In step 1, as soon as the reader thread completes reading the data into its buffer, it swaps the buffer

with the processing thread (step 2). This process continues until all the data is processed from the storage device.

B. Data Formatter

This component is responsible for reading the pre-fetched data, and converting it to the required format. The data is composed of sampled-time (microseconds) and corresponding current values (nanoamperes). The data is read in floating-point format, which is then converted into the integer format for efficient processing. The time is scaled to microseconds by first multiplying with 10^6 and then converted to integers, while the current data is converted to integers without scaling. The integer data provides sufficient precision for the time and the current data. When a sufficient amount of data (at least 32 MB) is formatted, it is handed over to the GPU Manager component for processing (pulse-detection) on the GPU (step 3 in Fig 2).

C. GPU Manager

The GPU manager transfers the data from the host memory to the GPU memory, launches the GPU kernel, and copies the data back from the GPU to host memory (step 4a, 4b, and 4c in Fig 2). The GPU Manager uses multiple streams to transfer data from the host memory to the device memory to overlap data transfer (I/O) with kernel execution (step 4a). Within a kernel, the input data is divided into equal size chunks which are mapped to the parallel processing threads (step 4b). Normally GPU memory is smaller than the host memory. The system can stage different size of data (32MB to 800MB) from the host memory into the GPU memory. We get the size of the GPU memory using CUDA API, and use 80% of the GPU memory for input data. The remaining 20% GPU memory is reserved for the working and the output buffers.

Based on the size of input buffer, we compute the size of the input data for each thread. For example, using 16K threads (256 threads per block and 64 blocks); input data size of 180MB, chunk size per thread is 11250 items. All threads concurrently operate on the assigned chunk to detect the pulses. Since pulses

in the input data are randomly distributed, some threads detect more pulses than others while some threads do not detect any pulses. We assigned an equal number of output slots to each thread. A thread that does not detect any pulses in its chunk stores only zeros in its output memory. The attributes of pulses (width, amplitude) detected by each thread in its chunk are accumulated in the output memory, which is part of the local memory on the device. The results are transferred to the host memory as soon as they are accumulated from a given stream (step 4c). When the results from all the threads are copied at the host memory, the CPU at the host machine merges the results to generate the consolidated result.

D. Result Analyzer

This component analyzes the detected pulse information (metadata) from the GPU. At this stage the resulting data is not in the exact format to be delivered to the user. The Result Formatter post-processes the information and generates the scatter plot of the detected pulses, which is step 5 in Fig 2.

Since the pulses detected are much smaller in number as compared to the input measured data, therefore post-processing of the result is performed on the CPU rather than on the GPU. This pulse information is further processed to extract only the required features i.e., width and peak of each detected pulse by removing zeros from the result. This information is then used to get the scatter plot of the pulses with width (microseconds) on x-axis, and its peak (nanoamperes) on y-axis as shown in Fig 4 in the results section. This scatter plot is delivered to the user for further analysis.

3.1.4. Results

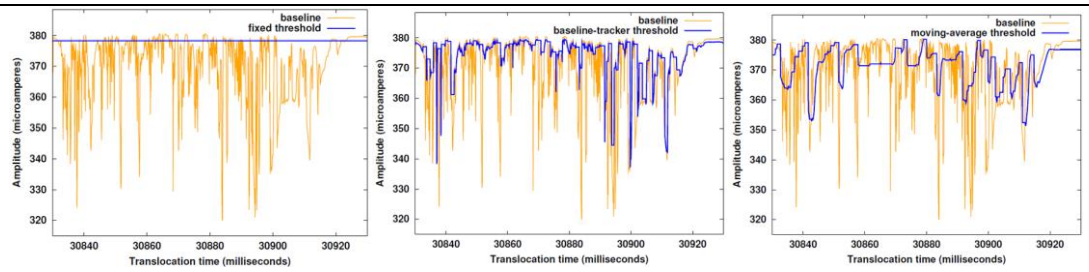
The designed system is measured by two key aspects; the algorithmic performance (in terms of pulse-detection), and the system performance (in terms of processing speed) for data generated in real time. For evaluating the algorithms' detection capability, we show the tracking of different threshold techniques for the given data with a varying baseline. We observe that a fixed threshold technique does not keep track of the data with a varying baseline, while with dynamic threshold technique; the threshold adapts itself to the changing baseline.

For CPU implementation, Intel Core i3 CPU consisting of 2 cores, each operating at 1.2 GHz, with 3 GB of RAM (computer memory) is used. This machine is used for both: CPU only sequential code and CPU coupled with GPU code. For the parallel implementation a CUDA-enabled GeForce 310M GPU with 16 cores clocked at 1.5 GHz and with 1 GB GPU (device) memory is used.

From system's performance point of view, the sampling rate is tested with different data sizes generated in real-time to measure the scalability of the system.

A. Algorithms Performance Measurements

The data is sampled with 75 microseconds and has highly time varying baseline. The baseline is actually the measured data from solid-state pores. Fig 3 shows the results obtained by using the fixed threshold technique Fig 3 (a), and the baseline-tracker technique Fig 3 (b). This is shown that the static technique can't keep track of the baseline, when it's highly varying, while baseline-tracker can still keep track of the



(a) (b) (c)

Fig 3. Shows the tracking of different threshold techniques to a noisy data. Part (a) captures fixed threshold technique, Part (b) captures the baseline-tracker technique, and Part(c) shows the tracking of moving average technique.

baseline. In moving average technique once the average is calculated, on w number of points, the detection process starts from $w+1^{st}$ point instead as shown in Fig 3 (c).

B. System Output Measurements

The output of all the three techniques applied to the input data are shown in the Fig 4. The data used is sampled with 2.2 microseconds and has relatively stable baseline than the data sample with 75

microseconds. Therefore, the pulses detected by fixed threshold technique shows a better overlap with the manual-detection, but also outputs noise Fig 4 (a). Baseline-tracker method detects pulses without noise but the detected pulses are more scattered as compared to the manual approach Fig 4 (b). The moving average approach has the best overlap with the manual approach with only 8.1% difference in the output Fig 4 (c).

For static and dynamic approaches the threshold is kept 1200 nanoamperes below the baseline, which detects the pulses more accurately as compared to other levels of threshold. Equivalent tolerance for moving average technique is 2.3%, but we have used 2.8% with which we have found better overlap with the manual detection.

C. System (GPU) Performance Measurements

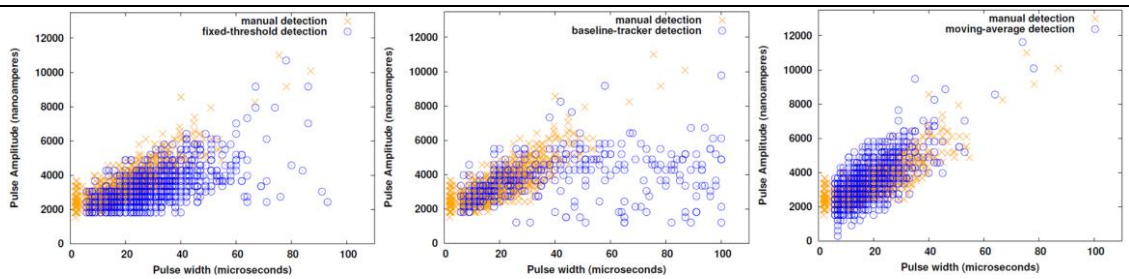
The GPU- and CPU-execution times for fixed, baseline-tracker and the moving average method are shown in Table 1 (a) – (c).

The temporal performance is measured for data size varying from 110 MB to 2.7 GB. The execution time is observed to be proportional to the size of data. Table 1d shows GPU time and the overall system (end-to-end) time that the execution time increases in proportion to the increase in the input data size. This shows that our system is scalable with different input sizes at the host and device levels.

This is observed that the data transfer time in the static threshold and baseline-tracker techniques is almost equal to the computation time, while in the case of the moving average technique more work is done per thread so the data transfer is still same as that of the other two techniques (static and baseline-tracker method), but computation part takes more time. Part of the reason for the greater computation time is that every CUDA thread initially loads its window with fresh data points from the assigned chunk rather than just adding the next data point and subtracting the first data point in the window which is the case for rest of the data points in the chunk.

3.2. Classification of Biological Targets

The classification of bio-targets entails machine learning algorithms, such as supervised learning techniques that learn from known patterns and then, classify the unknown based on the known patterns. However, the unsupervised learning techniques classify patterns without using known patterns. *k*-Nearest



(a) (b) (c)

Fig 4. Shows the comparison of results from automated techniques to the known results (a) shows output of static threshold technique to the known results, however, also detects noisy pulse as false positives, (b) shows the output of the baseline-tracker, which detected pulses with larger width, (c) shows the output of moving average technique, which closely matches to the known results, and differs only by 8.1%

Table 1. Execution time of GPU-kernel (with data-transfer time) and CPU for input data varied from 4 million points to 120 million points. **a** Fixed threshold technique **b** Baseline-tracker threshold technique. **c** Moving average Technique. **d** *Scalability:* Execution time with different input sizes for the moving average algorithm on GPU, and overall system time.

System-Input (GB)	**GPU-Input (MB)**	**GPU-kernel + Data-transfer time (msec)**	**CPU-time (msec)**
0.11 (4 M points)	32	25	28.93
0.54 (20 M points)	180	125.32	145.39
1.1 (40 M points)	320	250.74	295.02
1.7 (60 M points)	480	376.16	413.78
2.2 (80 M points)	640	501.43	506.4
2.7 (100 M points)	800	626.86	669.04

(a)

System-Input (GB)	**GPU-Input (MB)**	**GPU-kernel+Data-transfer time (msec)**	**CPU-time (msec)**
0.11 (4 M points)	32	24.73	25.35
0.54 (20 M points)	180	123.56	147.57

1.1 (40 M points)	320	239.67	287.02
1.7 (60 M points)	480	366.93	428.13
2.2 (80 M points)	640	492.12	522.44
2.7 (100 M points)	800	612.35	675.87

(b)

System-Input (GB)	**GPU-Input (MB)**	**GPU-kernel+Data-transfer time (msec)**	**CPU-time (msec)**
0.11 (4 M points)	32	69.14	31.26
0.54 (20 M points)	180	345.68	156.29
1.1 (40 M points)	320	690.95	315.57
1.7 (60 M points)	480	1036.63	476.24
2.2 (80 M points)	640	1382.21	640.21
2.7 (100 M points)	800	1727.75	810.42

(c)

System-Input (GB)	**GPU time (msec)**	**System time (seconds)**	
		Cold Buffer-cache Time	**Hot Buffer-cache time**
0.11	55.9	2.43	0.558
0.55	278	9.3	2.55
1.1	555.7	21.64	4.93
1.65	834.9	28.5	7.36
2.2	1115	42.2, 31	9.651
2.7	1394	54.5, 51.4	13.1

(d)

Neighbor algorithm was chosen because of its simplicity and quickly learning capability to learn from the known data and then, evaluate the new data based on the features of the known data. Additionally, *k*-Means algorithm, an unsupervised technique was used to classify the data into known patterns. The subsections capture the implementation and application of *k*-Nearest Neighbor and *k*-Means to the biological data collected from Solid-State micropores.

3.2.1. Feature Computation

This module takes input from the pulse-detector as shown in Fig 6 (a-c) and gives useful features to the next module on GPU for further pattern-mining captured as in Fig 6 (d). The pulse width and amplitude alone are insufficient to visualize the clusters from different cell types due to the overlap seen in 2D scatter plots. Therefore, features pertaining to the *morphology* (width, amplitude), *geometry* (falling slope and rising slope) as well as *statistics* (mean, and standard deviation) per pulse are computed. *Hybrid feature* as the average of mean and standard-deviation of each pulse are also computed to facilitate the visualization.

Pulse and its features:

When the signal (current) falls below the given threshold of baseline current, and reverts back to its original level, all the current values $(p_1, p_{2...}\ p_n)$ included in this dip form pulse. The duration between the start of the pulse and end of the pulse is the pulse width $(t_k - t_1)$, where $(t_1, t_{2...}\ t_k)$ are the corresponding time instances at which the pulse current values were sampled. However, the pulse amplitude is the minimum value among the recorded values of the pulse i.e. $p_{\min} = \min_{i=1} k\{p_i\}$. *Rising slope* is the measure of steepness of the pulse, $p_{\min}$ to p_k i.e., $m_{rise} = \frac{p_k - p_{\min}}{t_k - t_{\min}}$. However, *falling slope* measures how abruptly the pulse falls towards its minimum and is defined as: $m_{fall} = \frac{p_{\min} - p_i}{t_{\min} - t_i}$. In case the numerator becomes zero (very rare) is also taken care of so that the system does not fall back into the halted state. Consequently, the system doesn't record such a pulse. The *mean* is simply the average of the recorded values per pulse, $\mu = \frac{\sum_{i=1}^{k} p_i}{k}$, and the *standard-deviation* per pulse is given by:

$s = \frac{\sqrt{\sum_{i=1}^{k}(p_i - \mu)^2}}{k}$. The *hybrid feature* of a pulse is computed as the average of mean and standard-deviation i.e., $\frac{\mu + s}{2}$.

The abovementioned features are further used not only to visualize the natural clusters of different cell types, but also to separate them out with the least amount of overlap in them. Pulse width, amplitude, and mean are the best features to discriminate the clusters using 3D scatter plots, in addition to the width, amplitude and standard-deviation. Unfortunately, the scatter plot still results in 50% overlap between WBCs cluster and cancer cluster. The ultimate goal is to minimize this overlap as much as possible. However, with little more visual inspection *width, amplitude*, and *hybrid feature* are found to completely remove the overlap between WBCs and cancer cluster.

3.2.2. *k*-Nearest Neighbor Machine Learning

This algorithm works mainly in three steps. First step is to find Euclidean distance between each test sample to every training sample, which yields an $n \times m$ matrix, where *n* is the total number of test samples, and *m* is the total number of training samples. Second step is the sorting of all training samples to each test sample, to get a row-wise sorted matrix. Finally, the assignment of test sample is done to the class with maximum proportion in *k* selected training samples. These steps are shown in Fig. 5, separated by *synchronization* barriers to guarantee the completion of previous step before proceeding to the next step. In the following we present the mathematical formulation of the aforementioned steps.

Calculating Euclidean Distance: Given two points $i = (i_1, i_2 ... i_n)$ and $j = (j_1, j_2 ... j_n)$, the 3-space Euclidean distance is given by:

$$D_{ij} = D_{ji} = \sqrt{[(i_1 - j_1)^2 + (i_2 - j_2)^2 + ... + (i_n - j_n)^2]} \quad (7)$$

In our case, *i* and *j* are the indices of the samples in A and B, which are the *test* and *training* sets, respectively. The goal is to find distance from every sample in test (vertex) to every sample in training (edge) set i.e., *test (0, 1… n)*, and *training (0, 1… m)*. This naturally evolves into fully connected *bipartite graph* [96]. The algorithmic complexity becomes $O(|A|*|B|)$, where $|A|$ and $|B|$ is the cardinality of A and B i.e., *n* and *m*, respectively, and the running time is $O(m \times n)$.

To achieve the task of distance computation in parallel, we distribute the workload among threads such that each thread finds one of the $n \times m$ distances. The indices per *test* and *training* samples are computed as: $test_{id} = thread_{id} / m$, and $training_{id} = thread_{id} \% m$, where $thread_{id} = (0,1,...,n \times m)$ in order to make sure that the distance from each test sample to every training sample is computed – employs per thread per distance computation. Using 3D, i.e., *width*, *amplitude* and *hybrid feature*, the Euclidean distance in 3D space between i^{th} test sample ($test_i$) and j^{th} training sample ($training_j$) is given by:

$$D_{ij} = \sqrt{\begin{array}{l}[(test_i.width - training_j.width)^2 + (test_i.amplitude - training_j.amplitude)^2 + \\ (test_i.hybrid - training_j.hybrid)^2]\end{array}} \tag{8}$$

Sorting distance matrix: Once we compute distances from each test sample to every training sample, we get an $n \times m$ distance matrix, such that d_{ij} in the distance matrix is the distance between i^{th} test sample to the jth training sample as shown below:

$$D_{ij} = \begin{bmatrix} d_{11}' & d_{12}' & \dots & d_{1m}' \\ d_{21}'' & d_{22}'' & \dots & d_{2m}'' \\ \dots & \dots & \dots & \dots \\ d_{n1}''' & d_{n2}''' & \dots & d_{nm}''' \end{bmatrix} \tag{9}$$

Each row is sorted in the ascending order with the closest training samples d_{i1} next to its respective test sample *i*, and subsequently, the farthest training sample d_{im} at the end of the row, as given below.

$$D_{ij} = \begin{bmatrix} d_{11} & d_{12} & \dots & d_{1m} \\ d_{21} & d_{22} & \dots & d_{2m} \\ \dots & \dots & \dots & \dots \\ d_{n1} & d_{n2} & \dots & d_{nm} \end{bmatrix} \tag{10}$$

Decision in k neighborhood: Finally, we select first *k* training samples for each test sample and classify the test sample based on the maximum participation of training samples from a particular class.

$$D_{ij} = \begin{bmatrix} d_{11} & d_{12} & \dots & d_{1k} \\ d_{21} & d_{22} & \dots & d_{2k} \\ \dots & \dots & \dots & \dots \\ d_{n1} & d_{n2} & \dots & d_{nk} \end{bmatrix} \tag{11}$$

$$P_{ij} = \begin{bmatrix} prop_{11} & prop_{12} & \dots & prop_{1c} \\ prop_{21} & prop_{22} & \dots & prop_{2c} \\ \dots & \dots & \dots & \dots \\ prop_{n1} & prop_{n2} & \dots & prop_{nc} \end{bmatrix} \tag{12}$$

The computed proportions from all participating classes are used to compute the maximum proportion, such that for each test sample, $\max_{j=1}^{c}\{prop_{ij}\}\forall i$. The test sample is assigned with the color of the class that is occurring with the largest proportion in its *k* neighborhood.

A. *k*-Nearest Neighbor Classifier

Once the features per pulse are computed, these are fed to the pulse classifier to classify these into their respective clusters. The pulse classifier use machine-learning technique including *k-Nearest Neighbor* to detect cancer cluster and deliver results to the visualization module. The *k-Nearest Neighbor* groups the overall data into the training and test samples, and based on the training samples, the test samples are classified. The pulse-classifier can be executed either on the CPU or onto GPU selected by the user. Normally, the GPU memory is smaller than the main memory. However, the detected pulses are relatively smaller in number and are easily accommodated in GPU memory. The GPU uses multiple threads in order to run machine-learning algorithm in parallel. The parallel machine-learning algorithm is executed on GPU, as explained below.

The test samples (*n*) are classified using *k-Nearest Neighbor* classification based on different percentages of training samples (*m*). Generally, larger proportion of training samples results in better classification of the test samples. First of all, Euclidean distance for each test sample to every training sample is computed as explained before. This results in $n \times m$ matrix. To compute each distance in parallel at finer granularity, threads equal to $n \times m$ are launched such that each thread is responsible to compute distance per pair of a test and training sample. However, the amount of training samples and the remaining test samples impact the total amount of threads responsible for computing Euclidean distance. That is, the total number of threads launched could be as minimum as $nx1$, or $1xm$ when the proportion of training samples is huge ($n >> 1$) and vice versa. The number of threads could be as maximum as $n \times m$, when 50% of data is used as training samples.

In our case, 50% proportion requires approximately 130x131 = 17161 threads, accomplished with a configuration of 67 blocks and 256 threads per block. The output of the Euclidean-distance kernel is distance-matrix containing test samples as rows, and training samples as the columns. The distance-matrix is fed to the sorting kernel in order to sort the matrix row-wise, i.e., to achieve sorted training samples in ascending order for each test sample based on its distance from the test sample. Finally, the

sorted distance-matrix is the input to the decision-making kernel. This kernel selects the first *k* sorted training samples $(m_1, m_2 \ldots m_k)$ for each test sample and computes the proportions of each class $(prop_1, prop_2 \ldots prop_k)$ in them. The maximum proportion out of these is calculated such that $prop_{max} = max_{i=1}^{k}\{prop_i\}$. The test sample is then assigned to the class *i*, i.e., the class with the largest proportion in its *k* neighborhood.

B. Visualization

Visualization provides statistical information effectively in abstract form [97]. Thus, the results are organized into 3D scatter plots for pattern analysis of cancer cell and ultimately disease. Humans can better visualize up to three dimensions, and the task becomes tedious beyond three dimensions due to the increased number of features leading to the *curse of dimensionality*. Consider 2D scatter plots, there exists 30 different ways 6×5 of selecting 2 out of total 6 pulse features. This becomes more critical in case of 3D scatter plots. That is, there are 120 possible ways $6 \times 5 \times 4$ for selecting 3 out of 6. To reduce this manual overhead, we select features without repetition such that the order does not matter. Therefore, selecting 3 out of 6, the total number of possible scatter plots leads to 6!/(3!(6-3)!) = 20. To further reduce the number of scatter plots and non-significant features, such as falling slope and rising slope are ignored. These do not contribute much to separate out different clusters any ways. Given a total of 5 features and selecting 3 out of these, the total number of combinations (where order doesn't matter), comes out to be5!/(3!(5-3)!) = 10. Finally, only two combinations out of 10 ways, i.e., (*width, amplitude, mean*) and (*width, amplitude, standard deviation*) are found the best for discriminating the clusters of RBCs, WBCs, and cancer. However, to further separate out clusters, hybrid feature based on the average of mean and standard deviation per pulse is used. These features including *width, amplitude* and *hybrid feature* completely separated the cancer cells from WBCs and RBCs.

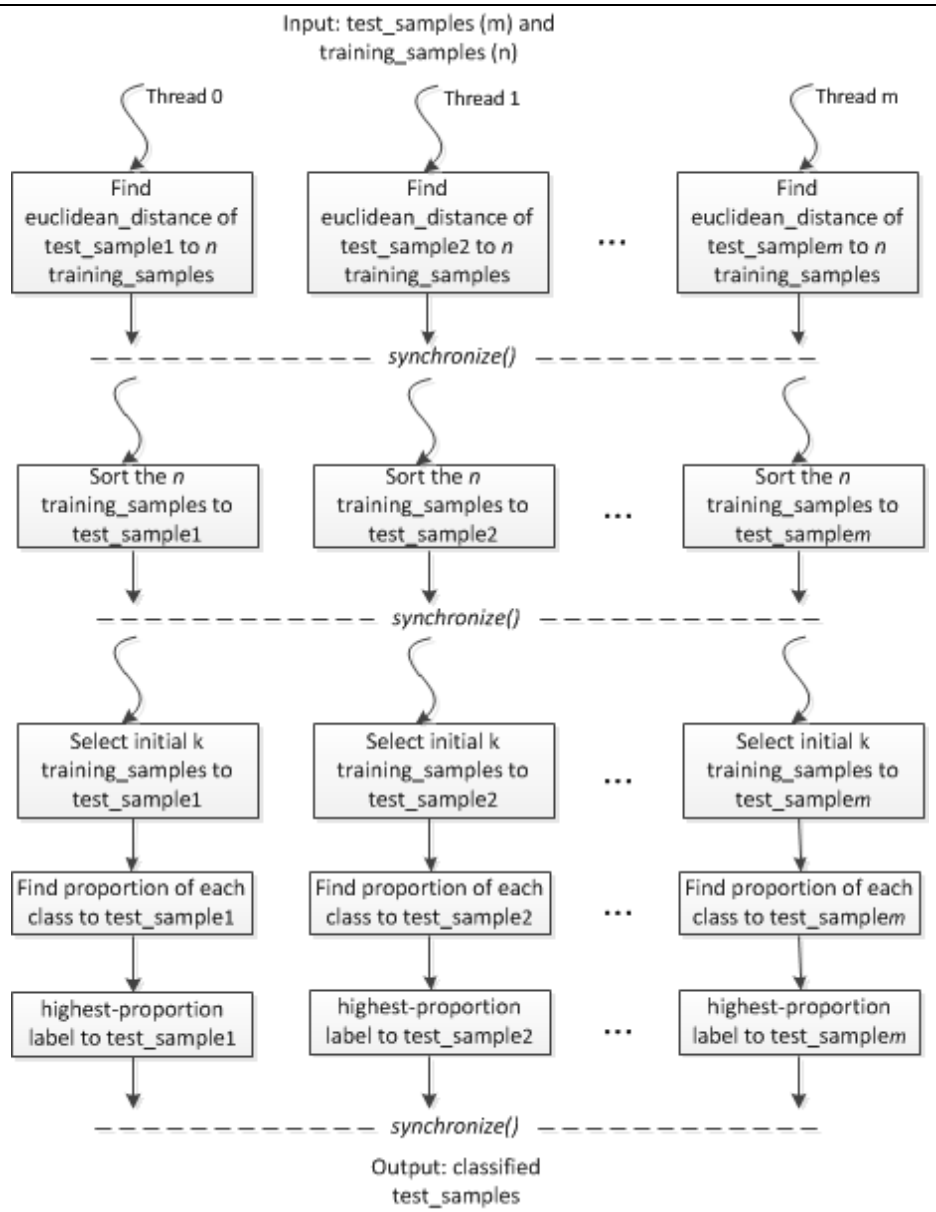


Fig 5. Block diagram of *k-Nearest Neighbor* algorithm. The first step finds the Euclidean distance from all test samples to the training samples. The next step sorts the distances row-wise such that every test sample gets training samples to it in ascending order, with the closest first. Finally, k training samples are selected from each set and based on the maximum proportion of class, the test sample is classified to that class.

C. Results

Experimental Setup

i7-based GPU setup: The system used for experimental setup was i7-based CPU coupled with NVIDIA Quadro FX 580. The CPU was equipped with 4 cores clocked at 1.6 GHz and total memory of 6 GB. The GPU used in our setup had 500 MB global memory, 64 KB constant memory, 16 KB per block shared memory and 8K of registers. Also, the GPU hosted a total of 32 cores, that is, 4 multiprocessors (MP), with 8 cores per MP, clocked at 1.12 GHz.

CUDA: With the given hardware and CUDA capability 1.1, the execution configuration could have a maximum of 512 threads per block. To achieve sorting in the case of *k-Nearest Neighbor* algorithm on GPU, CUDA-based thrust library [98] was used. For CPU-based counter execution, quick sort was adopted provided by the C library [99]. To measure the execution of kernel functions within the machine-learning algorithm, built-in CUDA timer functions i.e., CUDA events including *cudaEventCreate(), cudaEventRecord()* followed by *cudaEventSynchronize()* were used. These made sure that previously issued CUDA events were recorded on the GPU. CUDA *syncthreads()* was used to attain the synchronization across all the thread blocks.

Pthreads: We used Pthread library [100] to implement *double buffering*. During our experiments, buffers of size 200,000 each were found optimal for our problem. The integer size on 64-bit Intel machine running Linux OS with gcc complier was 4 bytes. Given the time and current values in integers per sample contributed to 8 bytes per sample, overall results were limited to 1.6 MB per buffer.

Lines of code: We implement our system using C and CUDA with 349 lines of code for moving-average filering, 53 for feature extraction, and 600 for *k-Nearest Neighbor* algorithm.

Datasets: Three different bio-sets describing several profiles of RBCs, WBCs and cancer cells, were tested. This data was collected from the micropore experimental setup with a pore diameter of 12 micrometer [39]. These bio-sets contained current values in microamperes, sampled at a rate of 2.2 microseconds. The raw data occupied around 9 GB of the storage device, pertaining to 360 million

current values. After pre-fetching the data into main memory, and pre-processing the raw data into the integers, the total amount of data shrank down to 2.88 GB, resulting in 68% reduction of the total acquired raw data. This could easily fit in the available 6 GB main memory.

i. Detection Results

Empirically, pulses that had widths greater than four samples (approximately 9 microseconds), and amplitudes greater than 1000 nanoamperes, were only recorded. The moving-average filtering used a sampling window of 2000 samples to achieve better detection of pulses. This reduced the pre-processed data to a total of 261 pulses — 26 RBCs, 208 WBCs, and 27 cancer cells. We were able to detect cancer cells from the raw data with an accuracy of 70%.

Typically the standard-deviation (noise) of the acquired data from cancer cells, WBCs and RBCs was found 492, 499, 463 nanoamperes respectively. Therefore, cancer and WBCs requires the threshold away from the baseline, as compared to RBCs. Thus to detect cancer cells and WBCs threshold was set to 0.036. However, a threshold of 0.026 was chosen to precisely detect RBCs, as shown in Fig 6. The insets in Fig 6 (a) – (c), show typical pulses of each type in detail. The baseline shift and the tracking of moving average filtering were also captured in Fig 6 (a). Obviously, cancer cell pulses were larger as compared to the WBCs and RBCs. Also, RBCs pulses fluctuated and were unstable as compared to the cancer and

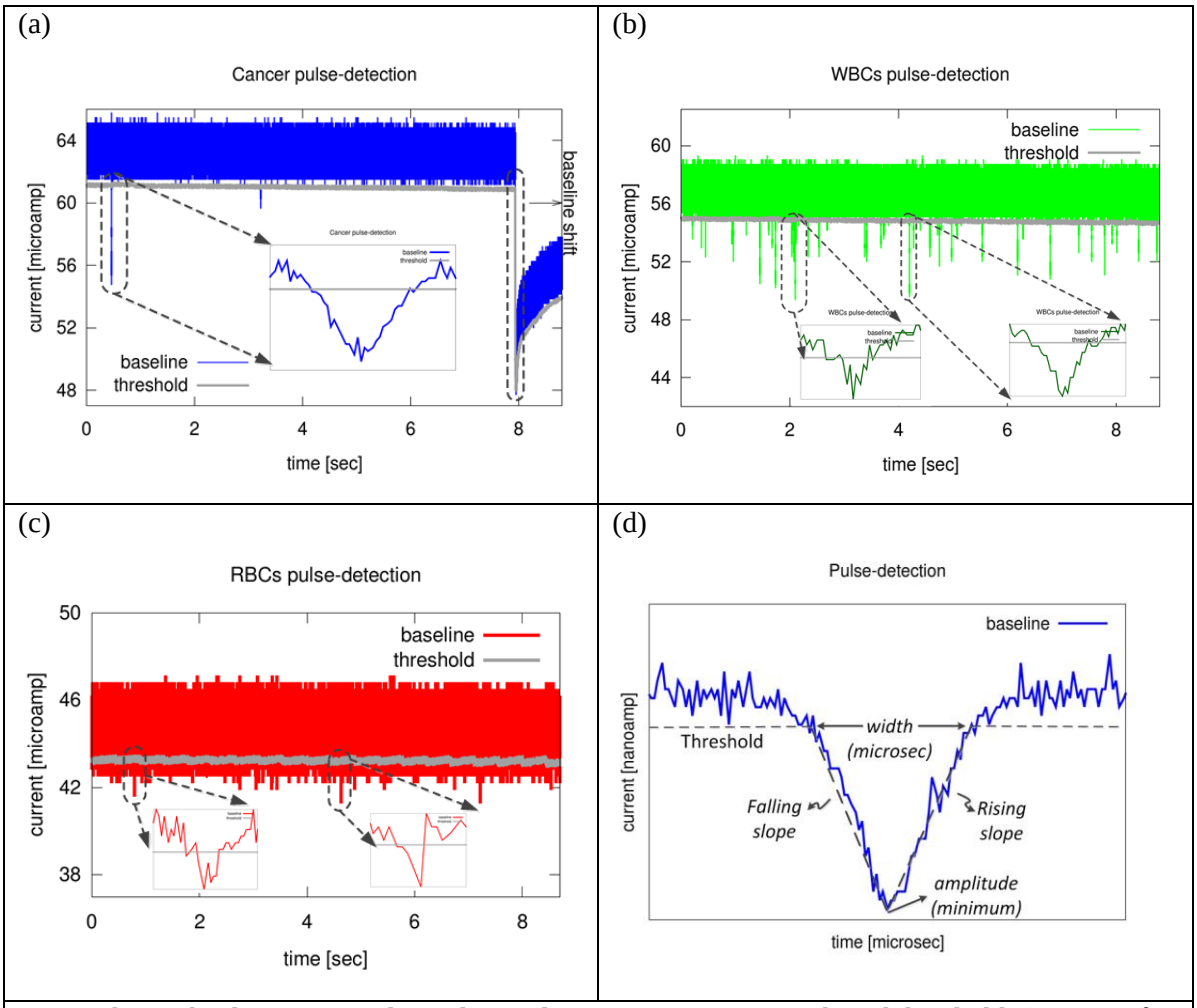


Fig 6. Shows the detection results with gray line as moving-average based threshold. (a) Data of Cancer cells with baseline shift acquired from solid-state micropore, (b) WBCs data collected from solid-state micropore, (c) Raw data of RBCs collected from solid-state micropore, (d) Typical pulse showing its width, amplitude (minimum), falling slope and rising slope.

WBC pulses. The two edges emerging out in the RBC pulses led to the insight that RBCs were clumped together when passed through the micropore – most probably due to their small size [39].

Additionally, features of the detected pulses were computed, such as pulse-width, pulse-amplitude, mean, standard deviation, falling slope and rising slope from the recorded samples per pulse. The width, amplitude, falling slope and rising slope were captured in Fig 6 (d). Pulse-width and pulse-amplitude were found statistically significant features in separating out the clusters. However, these

features still resulted in a 50% overlap of cancer cells with the WBC cluster. With further statistical analysis, we found that the average of mean and standard deviation (*hybrid feature*) when used as third dimension to the width and amplitude separated out the cancer cluster completely.

Therefore, pulse width, amplitude and hybrid feature were used for further analysis of different cluster types. The next section shows the statistical significance of the distinguishing features.

ii. Statistical Significance of Features

Table 2. The average of the features for the three different types of pulses

Cell types	Average pulse amplitude (microamp)	Average translocation time (microseconds)	Average of hybrid feature (microamp)	Average of falling slope	Average of rising slope
Cancer	8.7 ± 3.9	103.3 ± 74.9	23.4 ± 1.1	-497.2 ± 180.6	349.9 ± 138.5
WBCs	2.7 ± 1.1	21.7 ± 7.9	13.1 ± 0.6	-507.8 ± 207.3	412.3 ± 158.2
RBCs	2.3 ± 0.8	20.6 ± 9.1	11.3 ± 0.2	-470.2 ± 265.1	359.3 ± 149.3

We found average pulse amplitude greater for cancer cells and subsequently followed by the amplitude of WBCs and RBCs in Table 1. Such behavior was also seen for average translocation time, and hybrid feature. However, in case of falling slope, we observed larger average for WBCs than cancer cells while in the case of rising slope larger average of RBCs than cancer cells was seen. Such behavior of slopes did not make these useful for distinguishing the three types of human cells.

ANOVA Analysis of Pulse Features: Single-factor analysis of variance (ANOVA) was performed for the pulse translocation time (pulse-width), pulse-amplitude, falling slope, rising slope and hybrid features. We found p-value < 0.001 for pulse-width, pulse-amplitude, and hybrid feature, and therefore, these are significantly distinguishing features as compared to the slopes whose p-value > 0.05. Furthermore, the F-test models the variance between clusters to the variance within clusters. Greater F values results in compact clusters and hence more significant feature as compared to smaller values of F which means

scattered clusters resulting in overlap regions. The features are given below in the order of their significance with the highest F-value for hybrid feature:

$$F_{hybrid} > F_{pulse-amplitude} > F_{pulse-width} > F_{rising-slope} > F_{falling-slope}$$

3290.6 > 179.9 > 129.5 > 2.9 > 0.38

iii. Classification Results

The *k-Nearest Neighbor* is a supervised machine learning technique, where we first train the data by assigning labels to the samples, followed by the test samples which are classified based on the training data. The test sample is classified as the class that has maximum participation in its neighborhood *k*.

Different proportions i.e. 20%, 50% and 80% of the data were used as training data, and when *k* was kept 5 and 10, as shown in Fig 7. Accurate results were observed for larger amounts of training data. Furthermore, since cancer cluster was already separated out from the other two clusters using hybrid feature, otherwise overlapped, therefore, cancer cluster was detected 100% accurately in all cases except for 20% training data and when *k =10*, explained below.

a. Case with 20% Training Sample Data and k = 5 / k= 10:

In this case since the size of training data was small, therefore there was not enough information to classify the test data accurately. Note that 20% of RBCs, WBCs and cancer mean 5 RBCs, 42 WBCs, and 5 cancer cells respectively.

In case of *k = 10*, only one cancer cell was classified incorrectly as WBC as shown in Fig 7. However, all RBCs were classified as WBCs, again due to the small size of training data, which didn't capture enough information to allow RBCs to be classified accurately. Another reason for this misclassification was the large neighborhood as compared to *k = 5* shown in Fig 7, in which the proportion of WBCs was always larger, and the decision was in favor of WBCs even though the test point

was actually from RBCs, thus resulting in false positive. Such large proportion of WBCs in *k = 10* points was also responsible for the misclassification of one cancer cell.

b. With 50% Training Sample Data and k = 5 / k = 10:

In this case, cancer cluster was classified accurately. However, 50% of training sample meant half of the input data i.e. *detected pulses* were used as training, while half of them were used as test data.

For *k = 5*, all RBCs were classified correctly as shown in Fig 7. However, 25 WBCs were misclassified as RBCs. So there were 25 *false-positives* in RBC cluster, while 25 *false-negatives* in WBC cluster. In case of *k = 10*, 18 WBCs were misclassified as RBCs as shown in Fig 7 (c). This showed that 50% of the training data didn't have enough representatives of WBCs to classify these correctly. Furthermore, the significant overlap between RBCs and WBCs was also one reason for this misclassification.

c. Case for 80% Training Sample Data and k = 5 / k = 10:

Increasing the percentage of training sample data reduced the misclassification rate of WBCs. That is, in case of k = 5, only 2 WBCs were classified incorrectly as RBCs, as shown in Fig 7 (e). However, in the case of *k* = 10, we were able to classify the test data accurately for all types of cells captured in Fig 7 (f).

d. Accuracy of Classification:

The accuracy measures using confusion matrix for 50% and 80% of training data are shown in Table 3. The matrix captures the actual number of different cell types, and the predicted number of cells after classification, for k = 5 and k = 10, respectively. The total human cells were 261, out of which 26 were RBCs, 206 were WBCs and cancer cells were total 27. With an 80% of training samples from each cell type, the 20% of the test data was actually 6 RBCs, 42 WBCs and 6 cancer cells. However, 2 false-positive resulted in case of RBCs, stemming from the misclassification of WBCs as RBCs. Furthermore,

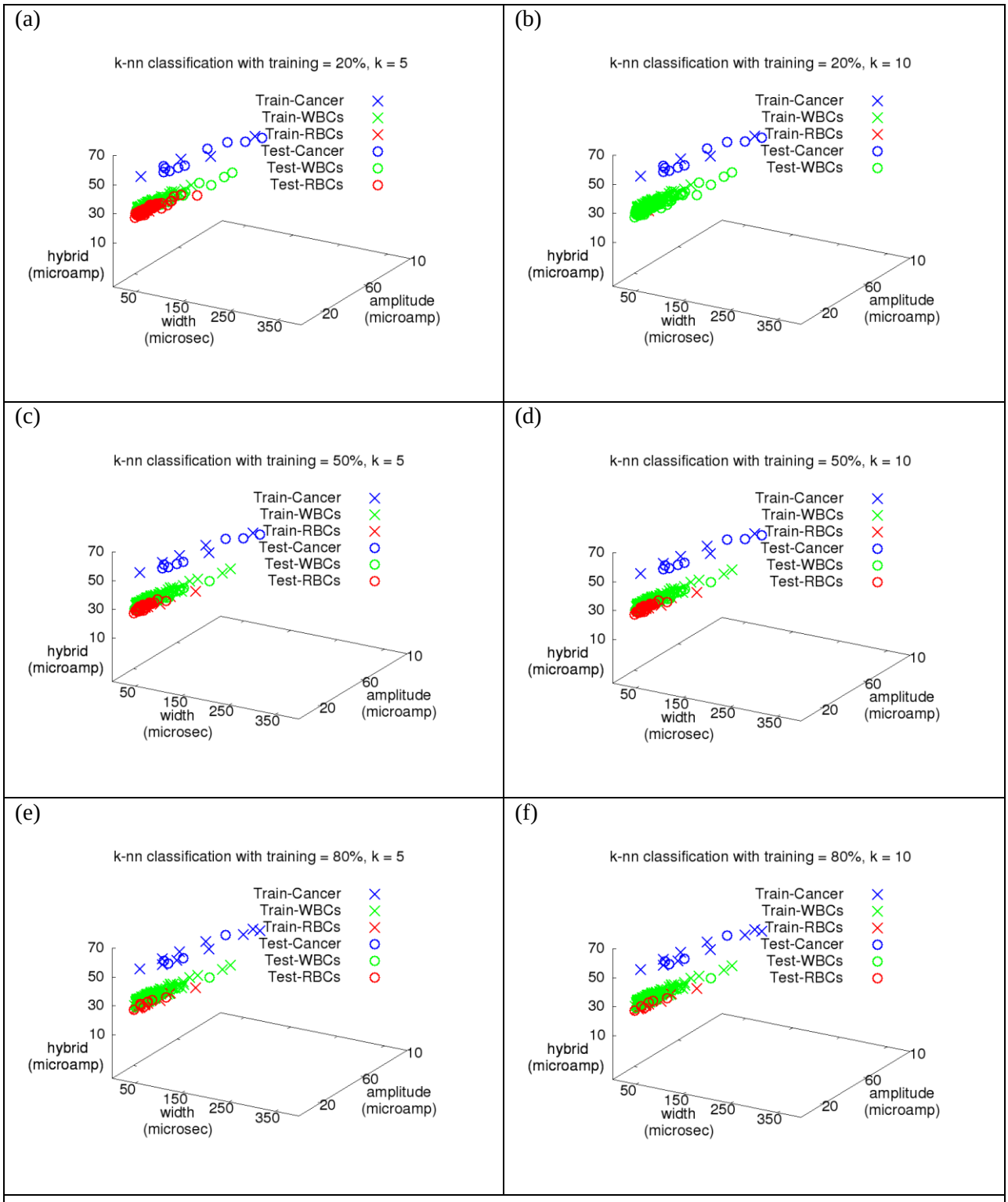


Fig 7. The results of *k-Nearest Neighbor* algorithm are shown with varying amount of training sample data. (a), (b) show cluster outcomes with training samples of 20% and *k*=5 and *k*=10 respectively. (c), (d) show cluster outcomes with training samples of 50% and *k*=5 and *k*=10 respectively. (e), (f) show cluster outcomes with training samples of 80% and *k*=5 and *k*=10 respectively.

Table 3. Confusion matrix showing accuracy of classification for different training sample-sizes (80%, 50%)

Training Sample=80%						
	k = 5			*k* = 10		
Actual Class	***Predicted class***					
	RBCs	**WBCs**	**Cancer**	**RBCs**	**WBCs**	**Cancer**
RBCs	6	2	0	6	0	0
WBCs	0	40	0	0	42	0
Cancer	0	0	6	0	0	6
Training Sample=50%						
	k = 5			*k* = 10		
Actual Class	***Predicted class***					
	RBCs	**WBCs**	**Cancer**	**RBCs**	**WBCs**	**Cancer**
RBCs	13	25	0	13	18	0
WBCs	0	79	0	0	86	0
Cancer	0	0	14	0	0	14

in case of 50% training samples, the number of false-negatives in WBCs was 25 and 18, for *k* = 5, and *k* = 10, respectively, which were miss-classified as RBCs.

iv. Performance Results

This sub-section highlights the performance of algorithms on both CPU and GPU. The Euclidean distance computation resulted in up to 3X speedup on the GPU as compared to CPU (Table 4). The reason that sorting was slower on GPU is due to copying of data back and forth for each test sample. The decision-making part is implemented only on CPU due to branch divergence issues.

Table 4. Show the performance analysis of *k-Nearest Neighbor* algorithms on CPU vs. GPU. Execution time for the main steps of *k-Nearest Neighbor* algorithm is shown.

k-Nearest Neighbor			
GPU kernel execution time (msec) ***k = 5, 10***	***Train = 20% samples Test = 80% samples Matrix = 210x51 (threads, blocks)= (256, 42)***	***Train = 50% samples Test = 50% samples Matrix = 130x130 (threads, blocks) = (256, 67)***	***Train = 80% samples Test = 20% samples Matrix = 54x207 (threads, blocks) =(256, 44)***
euclid_dist	0.1427, 0.1428	0.217, 0.2216	0.165, 0.1668
sorting_with_keys	143.38, 143.5	113.67, 113.9	44.48, 42.25
CPU time (msec) ***k = 5, 10***	***Single thread execution***		
euclid_dist	0.53, 0.526	0.526, 0.53	0.54, 0.55
sorting_with_keys	1.15, 1.05	1.05, 1.06	1.43, 1.43

decision_making	0.245, 0.249	0.180, 0.184	0.134, 0.14

3.2.3. *k*-Means Pattern Mining

For implementing *k*-means on GPU, we do not require any training samples, therefore all the samples are considered at once to find the natural clusters i.e., the compact cloud of samples. To achieve this, first *k* random centroids are selected, and given samples are assigned to the centroids based on their affinity determined by the Euclidean distance. All the samples are assigned to their respective centroids which results in *k* clusters. The centroids are then updated by taking the average of the samples assigned per cluster. These steps are shown in Fig. 8, with the first step computing Euclidean distance, second finding minimum centroid to a given sample and assignment of sample to it, and finally updating the centroids.

Calculating Euclidean Distance: The task of finding Euclidean distance from all *N* samples in set A to every *k* centroid in set B, arises into a fully connected bipartite graph [96]. The algorithmic complexity comes out to be: $O(|A|*|B|)$, where $|A|$ and $|B|$ is the cardinality of A and B i.e., *N* and *k*, respectively. Therefore, the distance equation with the most distinguishing features i.e. *width*, *amplitude* and *hybrid* feature is given by:

$$D_{ij} = \sqrt{\begin{array}{l}[(sample_i.width - centroid_j.width)^2 + (sample_i.amplitude - centroid_j.amplitude)^2 + \\ (sample_i.hybrid - centroid_j.hybrid)^2]\end{array}} \tag{13}$$

This results in *N* x *k* matrix with *N* samples and *k* centroids.

$$D_{ij} = \begin{bmatrix} d_{11} & d_{12} & \dots & d_{1k} \\ d_{21} & d_{22} & \dots & d_{2k} \\ \dots & \dots & \dots & \dots \\ d_{N1} & d_{N2} & \dots & d_{Nk} \end{bmatrix} \tag{14}$$

Computing Minimum Distance Centroid: The minimum distance to each samples is computed such that

$\min_{j=1}^{c}\{d_{ij}\}\forall i$. Each sample is assigned to its minimum distance centroid. Once all the samples are grouped into their respective clusters.

Updating Centroids: The centroids are updated by computing the average of the samples assigned to each cluster: $sum_i = \sum_{i=1}^{N_i} sample_i$ implies that new centroid is: $c\_new_i = \frac{sum_i}{N_i}$, where *i = 1 to k*. The whole process is repeated until the centroids converge, i.e, $\forall |c_i - c\_new_i| \cong 0$.

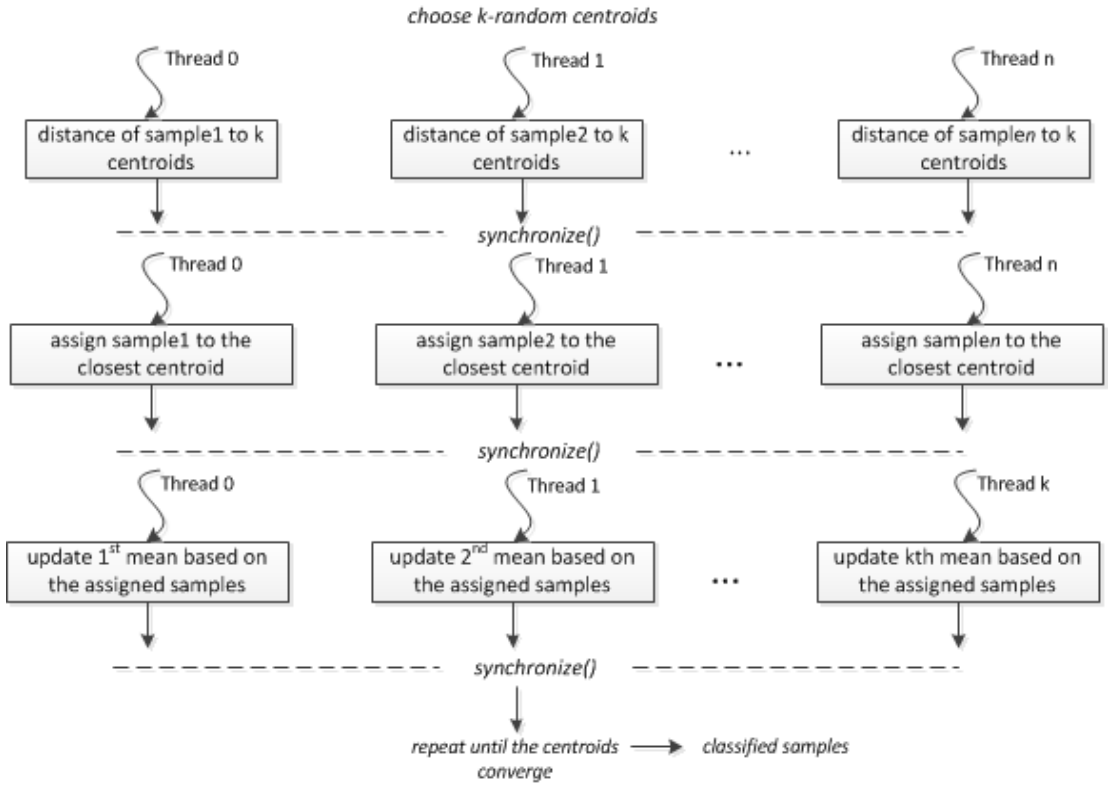


Fig. 8. Show block diagram of *k*-means mainly divided into three steps. The first step computes distances from each sample to all the *k* centroids. Next, the closest centroid is computed, and the sample is assigned to it. Finally, if not converged, the centroids are updated based on the samples assigned to it.

A. *k*-Means Classifier

Once the features per pulse are computed, they are fed to the pulse classifier to classify them into their respective clusters. The pulse classifier use pattern-mining technique including *k*-means to detect cancer cluster and deliver results to the visualization module.

The parallel clustering algorithm executed on GPU is explained below.

k-Means classifies the input samples into natural cluster. However, this technique is sensitive to the initial choice of clusters. To cope with this, we assigned random labels to all the samples and then *k*-samples were selected randomly as initial centroids to have an unbiased selection of the initial clusters. Additionally, the technique is also sensitive to the shape of clusters, and can only search for *hyper-spherical* clusters. The main reason for this problem is the use of Euclidean distance which works on the proximity of the samples in a given cluster. For instance, two rectangular-shaped clusters intersecting each other with its portions across the edges can be realized as part of the other cluster based on the notion of proximity of Euclidean distance. More sophisticated distance metrics based on the symmetry can tailor *k*-means algorithm to detect such clusters [101]. In our problem all the three clusters are naturally spherically shaped and are identified by 3-means. Nonetheless, 4, 5, and 6-means are able to identify cancer cluster explicitly, however, they mine further smaller clusters in RBCs and WBCs.

Input to *k*-Means algorithm consists of features of the detected pulses. This module launches Euclidean distance kernel to find the distance of each sample to the randomly selected *k* centroids. This results in $n \times k$ distance matrix, with rows the sample data and columns the centroids: $(c_1, c_2 .. c_k)$. Another kernel is launched to find the minimum distance out of the *k* distances for each sample i.e., $c_{\min} = \min_{i=1} k\{c_i\}$ and assign the color *i* of the closest centroid to the sample c_i. This reduces the distance matrix to $n \times 2$ with sample data along with the assigned color. Finally, centroids are updated based on the assigned samples to it. i.e., $c\_new_i = \sum_{i=1}^{sum_i} \frac{sample_i}{sum_i}$. The process is repeated until the results converge i.e., there is no or negligible difference between c_new_i and c_i. We use separate thread for each sample during the Euclidean-distance finder and min-distance finder kernel, but for updating centroids, the number of threads is equal to the number of centroids.

B. Results

i. Classification Results

The input data to *k*-means is detected pulses with no labels to them. Fortunately, the Cancer cluster is well-separated as compared to the RBCs and WBCs cluster, and thus forming a natural cluster in the data. *k*-Means is able to detect the Cancer cluster with any value of *k* starting with 2, as shown in Fig 9. Obviously, we see two clusters in the figure, the Cancer cluster and the other two clusters from the RBCs and WBCs. However, the Cancer cluster is not dense, but is uniform and well-separated from the rest of the lower cluster forming dense cloud. Therefore, Cancer cluster is detected naturally with different number of centroids, even with the number of centroids greater than 2. In the latter case, all the remaining centroids are concentrated in the lower cluster.

ii. Convergence Results

The Table 5 shows the initial choice of centroids, the number of samples initially found closer to the selected centroids, and later, the algorithm formed final centroids with the number of samples per cluster. The results show that in all cases Cancer cluster is converged to a centroid of 103, 8715, 44195 i.e., *width, amplitude,* and *hybrid feature* respectively, for all its 27 samples.

Different values of *k* were successful in detecting the cancer cluster. The results are shown for *k* = 2, 3, 4, 5, 6 and 7-means.

iii. Performance Results

This sub-section highlights the performance of algorithms on both CPU and GPU. In case of computing Euclidean distance for k-means on GPU, trend is observed that as the number of k increases, the GPU approaches the speed of CPU, such that for k = 7, we see improved performance on GPU instead. Part of the reason is the increase in the number of threads, in addition to the larger occupancy of the GPU.

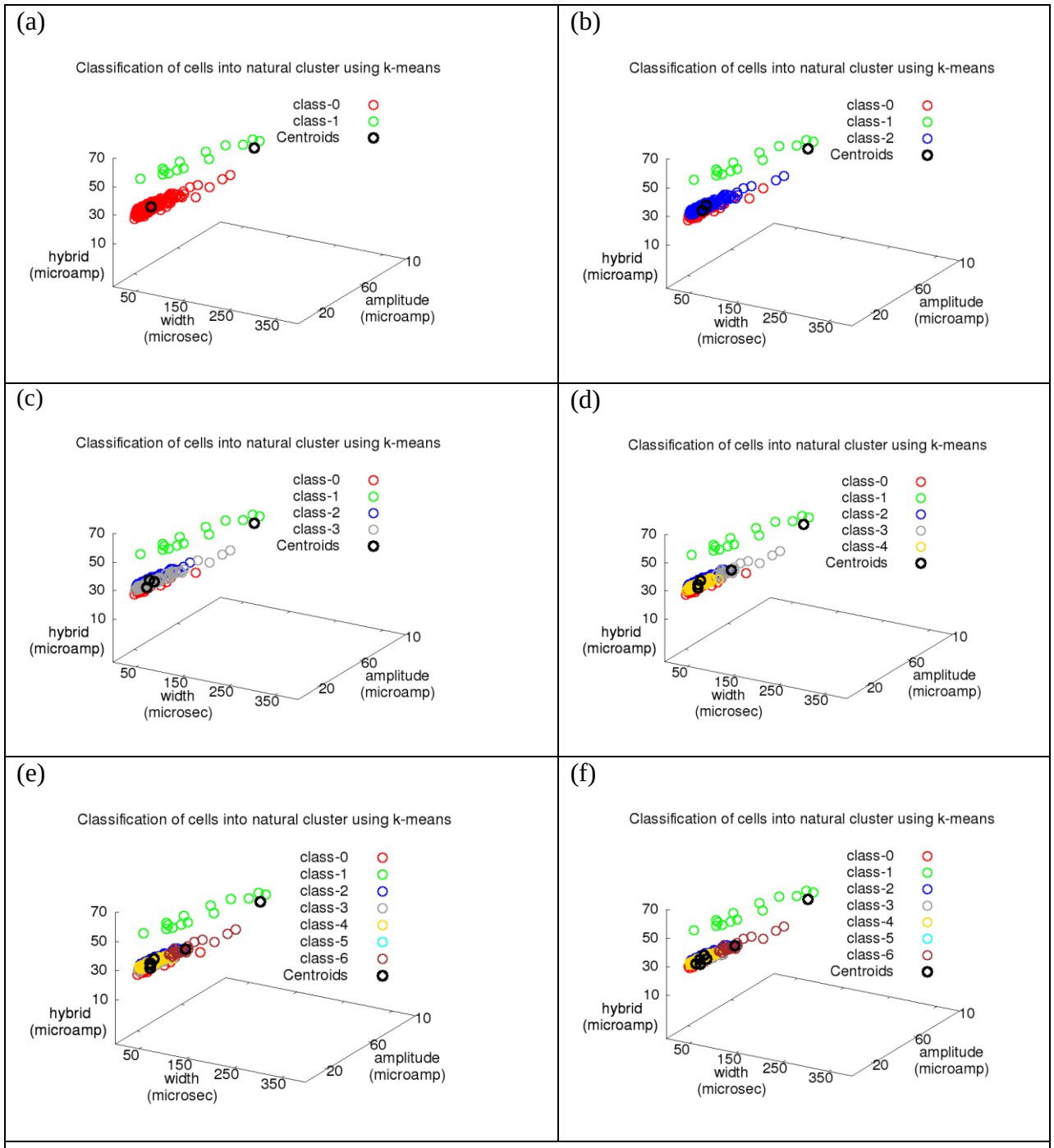


Fig 9. *k*-Means shows the detection of cancer as a natural cluster in the classified clusters stemming from the detected pulses. Different values of *k* starting from 2 to until 7 are tested, all of them detected Cancer cluster separately due to its larger dimensions. (a) *2*-means detected two clusters. The red cluster is inclined towards the bulk of the data, while the green cluster has detected cancer, (b) *3*-means detects the three clusters including cancer (green) cluster, (c) *4*-means, (d) *5*-means, (e) *6*-means, and (f) *7*-means detects cancer as a natural cluster, while the lower bulk of data forms 6 sub-clusters.

Table 5. Show *k-Means* with different number of initial and final centroids, and their convergence at a particular iteration.

k-Means	Initial centroids (*width, amplitude, hybrid*)	# samples per cluster	Final centroids (*width, amplitude, hybrid*)	# samples per cluster
2-means: convergence @ iteration # 4	1. 15, 2206, 23542	89	1. 21, 2690, 25078	234
	2. 31, 3164, 25908	172	2. 103, 8715,44195	27
3-means: convergence @ iteration # 5	1. 15, 2206, 23542	89	1. 21,2497,23550	96
	2. 31, 3164, 25908	81	2. 21, 2823,26141	138
	3. 18, 2630, 25817	91	3. 103, 8715,44195	27
4-means: convergence @ iteration # 8	1. 15, 2206, 23542	60	1. 20, 2308, 22538	39
	2. 31, 3164, 25908	79	2. 19, 2639, 26579	90
	3. 18, 2630, 25817	72	3. 23, 2875, 24735	105
	4. 18, 2470, 24849	50	4. 103, 8715,44195	27
5-means: convergence @ iteration # 12	1. 15, 2206, 23542	59	1. 20, 2311,22508	38
	2. 31, 3164, 25908	79	2. 20, 2345, 24749	93
	3. 18, 2630, 25817	61	3. 19, 2630, 26674	79
	4. 18, 2470, 24849	39	4. 36, 4819, 25168	24
	5. 22, 1672, 25192	23	5. 103, 8715,44195	27
6-means: convergence @ iteration # 9	1. 15, 2206, 23542	41	1. 20, 2326, 22030	26
	2. 31, 3164, 25908	27	2. 19, 2378, 25632	68
	3. 18, 2630, 25817	70	3. 20, 2380, 24120	65
	4. 18, 2470, 24849	52	4. 19, 2704, 27016	52
	5. 22, 1672, 25192	31	5. 36, 4867, 25213	23
	6. 23, 2790, 25152	40	6. 103, 8715,44195	27

7-means: convergence @ iteration # 10	1. 15, 2206, 23542	46	1. 18, 2025, 23897	34
	2. 31, 3164, 25908	72	2. 19, 2704, 27016	52
	3. 18, 2630, 25817	54	3. 23, 2773, 24365	34
	4. 18, 2470, 24849	30	4. 18, 2348, 25648	66
	5. 22, 1672, 25192	23	5. 36, 4867, 25213	23
	6. 23, 2790, 25152	21	6. 20, 2348, 21996	25
	7. 34, 3543, 21917	16	7. 103, 8715,44195	27

Table 6. Shows execution time for the main steps of *k*-Means algorithm CPU vs. GPU

k-Means, Total number of samples = 261						
GPU time (msec)	***k = 2***	***k = 3***	***k = 4***	***k = 5***	***k = 6***	***k = 7***
euclid_dist	0.046	0.045	0.044	0.043	0.042	0.041
min_dist	0.01	0.012	0.014	0.016	0.017	0.019
update_centroids	0.688	0.687	0.688	0.689	0.687	0.687
CPU time (msec)	***k = 2***	***k = 3***	***k = 4***	***k = 5***	***k = 6***	***k = 7***
euclid_dist	0.014	0.02	0.027	0.0324	0.038	0.045
min_dist	0.0032	0.003	0.0032	0.0033	.0034	0.0033
update_centroids	0.003	0.003	0.003	0.006	0.006	0.006

3.3. Two-Step Detection

The pulse is essentially a sequence of data points falling abruptly downwards and reverting back to the normal baseline forming a valley, with an acceptable base width. The goal of this work is to automatically detect the pulses with their distinguishing features in the noisy data generated from solid-state micropores.

Ideally the noise should be eliminated completely from the data in order to detect all the useful pulses efficiently, differentiating the rogue pulses (cancer cells) from the good pulses (healthy cells) based on their features. However, in practice there are many challenges posed by the sensor noise, noisy pulses, and subjectivity. Furthermore, the useful pulses are very sparse in the collected data, due to the high throughput data to capture the translocation behavior of the target cells in detail [92]. In order to mitigate the impact of noise, otherwise cannot be eliminated completely, and to efficiently detect the pulses, the designed technique is geared towards first reducing the noise and then detecting the pulses in the raw data [102]. Additionally, the automated technique needs to be faster to make it amenable for online monitoring setup. Towards this end, our design is composed of a three-step process.

- The raw data is converted to suitable format, that is, integers. Integer arithmetic avoids round-off errors, and also reduces the total size of acquired data.
- Second, the data is smoothed using moving average filtering to reduce the variation in the data and thus, the noise. However, the extent of smoothing is based on the size of sampling window used in moving-average filtering.
- To automatically select the threshold, it is computed from the mean and standard deviation of the smoothed data [34]. Finally, the threshold is used to detect the pulses falling below its level, along with their features, such as width and amplitude of the pulse.
- Integer processing, and recursive moving-average filtering [92, 103, 104] makes the technique faster, effective, and suitable for online monitoring and prognosis. Our technique efficiently detects the pulses and results are delivered in the form of scatter plot for further analysis towards diagnosis.

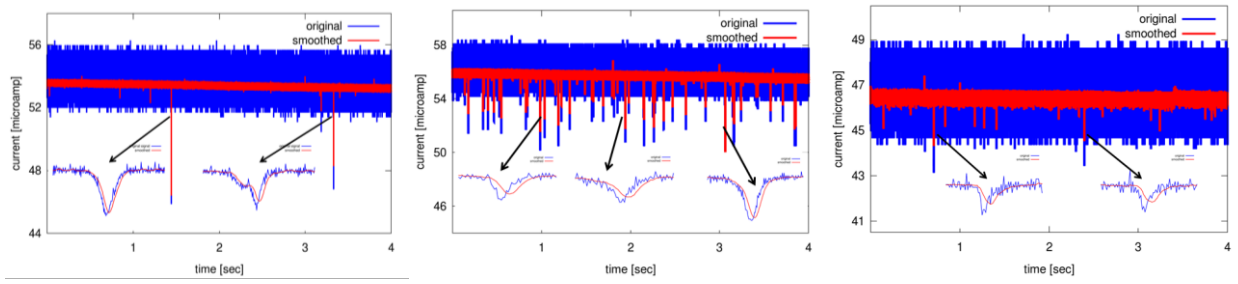


Fig 10. Smoothing of noisy data from different cell types and pulses in them. (a) Cancer pulse detection with highlighted pulses in insets. (b) White blood cell pulses with suppressed noise. The enhanced pulses are shown in the insets. (c) Red blood cell pulses with and without smoothing. RBCs have comparatively smaller pulses. The cancer pulses are larger as compared to other cell types. Also, the cancer pulses retain their shapes even after smoothing.

3.3.1. Methods

A. Pre-processing of Noisy Data

The acquired raw data was converted to integers for faster processing, and in addition integer arithmetic avoids round-off errors. Furthermore, this resulted in smaller size as compared to the raw data. Typical raw profile of cell translocation data comprised of 4 million samples, forming a 112 MB file. Using Intel i7-based CPU with gcc compiler running on top of linux operating system, the integer size of 4 bytes reduced 112 Megabytes (MB) to 57 MBs, almost half the size of original.

B. Smoothing

Low-pass filtering based on moving average smoothing got rid of high-frequency noise and left out low-frequency pulses. The cut-off frequency to decide which band of frequencies should be allowed to pass was inversely proportional on the size of sampling window used in moving-average filtering. Larger the size of sampling window, greater was the extent of smoothing, that is, cut-off frequency was low and only extremely low frequencies were allowed to pass. In contrast, higher cut-off frequency allowed fairly higher frequencies and thus, resulted in little smoothing. In that case, the size of sampling window was

smaller. Our approach tested moving-average for different size of sampling, that is, 5, 10 and 20. However, higher sampling size resulted in too much smoothing such that the original shape of the pulse was not preserved. We found sampling size of 5 to be optimal for our problem.

For a given time-series, $x_i, x_{i+1}, \ldots.$, the running sum with a sampling window of size k is given by:

$$running\_sum = (x_i, x_{i+1} \ldots x_{i+k}) \tag{15}$$

Subsequently, the moving average is computed by dividing the running sum over the size of sampling window.

$$moving\_average = running\_sum / k \tag{16}$$

To recursively compute the running average, the next value in the window was added, while the trail value in the window was dropped off. This way the *k-1* summations and one division were eliminated by only one addition, one subtraction, and one division. This resulted in improved performance, especially for large size of sampling window.

$$moving\_sum = moving\_sum + x_i - x_{i-k} \tag{17}$$

C. Compute threshold

Many techniques have been developed to detect peaks in time-series data[34]. Our approach was based on computing threshold from the statistics of the smoothed data. The mean for threshold was computed initially from the samples of size equal to that of moving-average window and was used then for further analysis of the data.

$$Threshold = mean - 4 * std\_deviation \tag{18}$$

However, the threshold was re-computed after every 200,000 samples. This way the threshold was trained with few samples, and then tested for thousands of samples. Additionally, the value of *k*, which was domain-specific, varied. For larger values of *k*, it was away from the smoothed baseline and could miss small useful pulses. In contrast, it was close to the baseline and could pick false pulses. In our case, we found $k=4$ as optimal. The threshold not only detected the pulse, but also computed its features including the width and amplitude.

D. Post-processing

The noisy pulses were eliminated with their width and amplitude not falling in the acceptable range, and only useful pulses were recorded. The final results in the form of scatter plots were delivered for analysis.

3.3.2. Results

A. Datasets

Datasets were collected for red blood cells (RBCs), white blood cells (WBCs), and cancer cells passed through solid-state micropore of 12 μm diameter. Overall data collected from a typical blood sample consisted of 90 profiles (around 10 GB raw data). A typical profile of cells data contained 4 million raw data points sampled at 2.2 microseconds. The current values were processed at the scale of nanoamperes, however, the results are shown in microamperes for the sake of simplicity.

B. Smoothing

Moving-average based smoothing eliminated high-frequency noise. The baseline noise was removed while the pulses were smoothly shaped, as shown in Fig 10 (a) – (c). However, cancer cells formed significantly larger pulses and thus were better shaped than WBCs. In contrast, RBCs were smaller and their pulses were simply bell-shaped. These couldn't be further categorized. The sample size of window was 10 in that case.

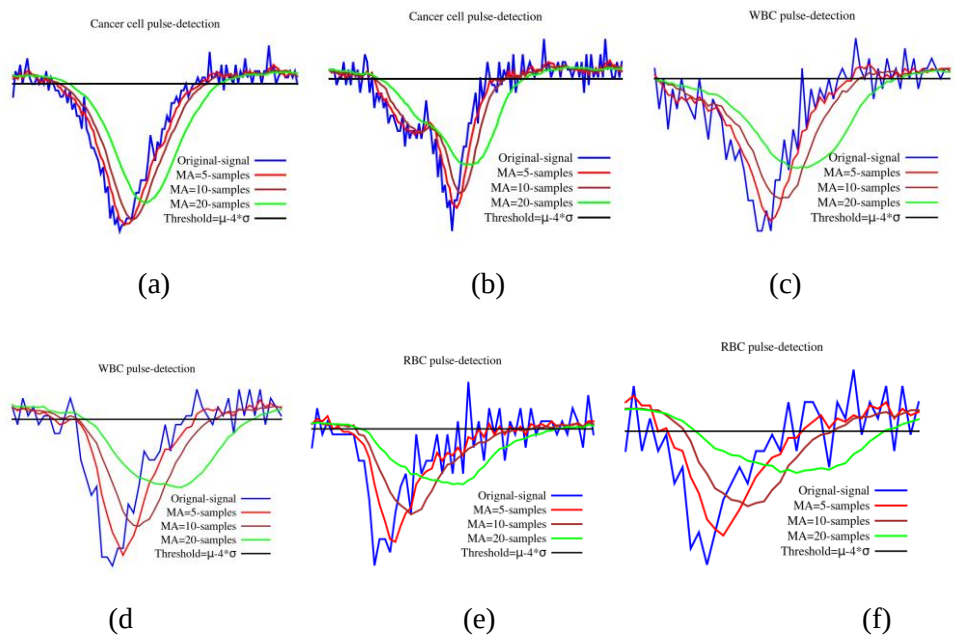


Fig 11. Typical pulses from each cell type shown along with the smoothing. Moving-average filtering is used for smoothing with sampling window size of 5, 10 and 20 for: (a) Typical cancer pulses, (b) Typical WBC pulses and (c) Typical RBC pulses.

Greater smoothing resulted in reduction of variation in the baseline to a greater extent. However, it also altered the shapes of the pulses significantly. This is clear from Fig 11 (a)-(c), where smoothing with a sample size of 5 resulted in better pulse shapes. However, the sample size of 10 and 20 didn't shape the pulse precisely but resulted in higher reduction of noise.

i. Impact of Threshold Selection

Different levels of threshold detected different number of pulses. Closer the threshold to the baseline, that is, *Threshold = mean – 3*standard-deviation*, resulted in the detection of noisy pulses (false positives). However, threshold away from the baseline, such as *Threshold = mean – 5 × standard-deviation*, missed even useful pulses (false negatives). Smoothing with a sample window of 5 followed by *Threshold = mean – 4 × standard-deviation* was found optimal for the detection of pulses from the three different

types of cells, captured in Fig. 12 (a), (b) and (c). The threshold was sufficiently below the smoothed baseline in all the three cases.

ii. Scatter Plots

Finally, the width and amplitude of the detected pulses were computed and dispalyed as scatter plots. Fig. 12 shows the pulses detected from all three different types of data.The resulted scatter plots showed the pulses spreaded horizontally, due to the fact that smoothing reduced the ampitude of the pulse due to eliminating the vertical fluctuations. It, however, increased the horizontal width of the pulses. The

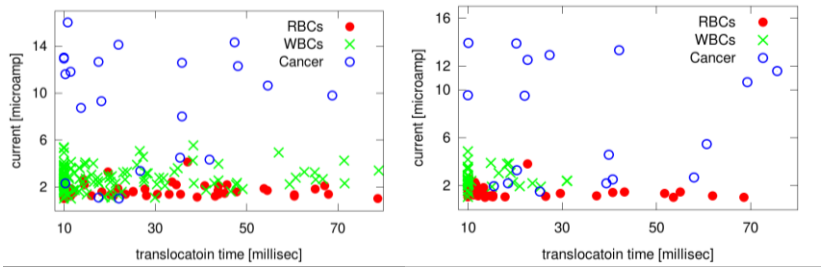


Fig 12. Smoothing of the pulses apparently reduces the amplitude of the pulses and increases the widths. Therefore, the detected pulses are scattered instead of forming compact clusters. Pulses detected in RBCs, WBCs and Cancer with (a) 5-sample moving average and Threshold = mean – 4 × standard-dev. (b) 7-samples moving average with a Threshold = mean – 4 × standard-dev. The larger smoothing (shown in b) eliminates the noisy pulses as compared to pulses of (a) with less amount of smoothing.

overlap was also seen among different kinds of pulses because some of the smaller pulses in cancer cells resembled closely to larger pulses of WBCs in their widths and amplitudes.

iii. Statistics

The variation removed by different levels of smoothing is delineated in Table 7. Greater smoothing, such as 20-sample moving average resulted in higher suppression of variation in the original signal, such as the standard deviation for RBCs was reduced from ±465.9 nanoamperes to ±65.5 nanoamperes, however this also deteriorated the shape of the pulse to greater extent.

Cancer pulses have greater pulse amplitude and width than the other cell types, as captured in Table 8. The p-value < 0.01 for translocation time shows that all cell types significantly differs from each other. In contrast, p-value > 0.01 for pulse amplitude shows that different cell types do not differ significantly from each other.

Table 7. Shows the impact of extent of smoothing to the variation in the original data

Types of cells → **Sampling Type ↓**	**RBCs**	**WBCs**	**Cancer Cells**
Original-signal	±465.9	±469.09	±492.1
5-sample smoothing	±152.72	±188.94	±181.03
10-sample smoothing	±93.16	±143.26	±135.87
20-sample smoothing	±65.5	±124.79	±116.18

Table 8. 5-sample moving-average and threshold = mean – 4 × standard-deviation

Pulse features	**Cancer**	**WBCs**	**RBCs**
Pulse amplitude average (microamp)	8.3 ± 3.89	3.14 ± 1.01	1.6 ± 5.4
Pulse width average (millisec)	30.1 ± 25.17	15.0 ± 90.3	27.8 ± 36.84

Automated pulse detection can greatly improve the online monitoring of cancer detection and diagnosis, and simplifies the clinical procedure, where raw data will be collected from many patients visiting per day. This can help physicians to change their strategies in order to combat deadly diseases like at very early stages.

3.4. Discussion

Our work has proposed a novel real-time system for the detection in millions of data points generated from biological targets when passed through a nanopore. However, the performance gap between I/O

operations and multi-core processing speed is a bottleneck for high-throughput data acquisition. Furthermore, the I/O and processing implementations are sequential in nature, the data is first read into the system, and then it is processed. To address this problem, Data Pre-fetcher (discussed in section 2.4.1) uses advanced I/O prefetching techniques, such as double buffering to overlap the I/O operations with the computation. This improves the system performance by reducing execution stalls. For further improving the system performance, integer processing is used, i.e., the raw data (floating-point numbers) is converted to integers, which enables efficient processing as compared to the floating-point computations. With such a design, a sampling rate of 2.3 million samples per second is achieved, as compared to the original rate of the electrical measurement setup which is 454 Kilo samples per second (5x improvement).

The GPU implementation for static and baseline-tracker technique results in 1.11x and 1.12x speed up over its counter CPU implementation. However, for moving average technique CPU is 2.2x faster than GPU. For static and baseline-tracker technique much time is spent in data transfer in addition to the computation time on GPUs. However, in case of moving average technique more time is spent in computation since every thread calculates its initial average on GPU which results in performance degradation, whereas on CPU the initial average is calculated only once. This shows that the GPU is faster but less clever than CPU due to its limited I/O and memory resources. The recent trends in CPUs (core i7) and GPUs (NVIDIA Tesla 2070) can result even in better speed-ups.

Furthermore, the input data is not parallel and consists of independent (background noise) and dependent data points (pulses); this hybrid nature of data poses an automation challenge on efficiently using GPU. To address this problem, the number of threads that can be used in our system is restricted. Therefore, if we increase beyond 16 K threads, the number of pulses detected is reduced because significant amount of pulses start to lie at the boundaries of data-chunks assigned to each thread and cannot be tracked across different data chunks.

The impact of automated threshold detection algorithms is evaluated in the cases of time-varying baseline and noisy data. The moving average detection technique is found closer to the manual detection technique with 8.1% difference in the output, less number of false alarms, and the detection is more tolerant to noise than fixed and baseline-tracker technique at the cost of computational overhead.

Hence the designed GPU-based real-time system is almost 5x faster than the electrical measurement setup. This work can have wide implications when hundreds of bio-sensors are measuring temporal data from the physiological or molecular interactions of living systems.

Chapter 4

Distributed Detection of Cancer Cells in High-Throughput Cellular Spike Streams

Detection and identification of important biological targets, such as DNA, proteins, and diseased human cells is crucial towards early disease diagnosis and prognosis. The key to differentiate healthy cells from the diseased cells is the biophysical properties that differ significantly. Micro and nanosystems, such as solid-state micropores and nanopores, can measure and translate these properties of human cells and DNA into electrical spikes to decode useful biological insights. Nonetheless, such approaches result in large data streams that are often plagued with inherit noise and baseline wanders. Moreover, the extant detection approaches are tedious, time-consuming, and error-prone, and there is no error-resilient software that can analyze large datasets instantly. The ability to effectively process and detect biological targets in larger datasets lies in the automated and accelerated data processing strategies using state-of-the-art distributed computing systems. To this end, we propose a distributed detection framework, which collects the raw data stream on a server node that then splits/distributes the data into segments across the worker nodes. Each node reduces noise in the assigned data segment using moving-average filtering, and detects the electric spikes by comparing them against a statistical threshold (based on the mean and standard deviation of the data), in a Single Program Multiple Data (SPMD) style. Our proposed framework enables the detection of cancer cells with an accuracy of 63% in a mixture of Cancer cells, Red Blood Cells (RBCs), and White Blood Cells (WBCs), and achieves a maximum speedup of 6X over a single-node machine by processing 10 gigabytes of raw data using an 8-node cluster in less than a minute.

Diseases such as cancer can be mitigated, if detected and treated at an early stage. Micro and nanoscale devices such as micropores and nanopores, enable the translocation of biological targets, such

as, DNA, proteins, and human cells at its finer granularity. These devices are tiny orifices in silicon-based membranes and the output is a current signal, measured in nanoamperes. Solid-state micropore is capable of electrically measuring the biophysical properties of human cells, when a blood sample is passed through it [23]. The passage of cells via such pores results in an interesting pattern (pulse) in the baseline current, which can be measured at a very high rate, e.g., 500,000 samples per second [92]. The pulse is essentially a sequence of temporal data samples that abruptly falls below and then reverts back to a normal baseline with an acceptable predefined time interval, i.e., pulse width. The pulse features, i.e., width and amplitude correspond to the translocation behavior and the extent to which the pore is blocked, under a constant potential. These features are crucial in discriminating the diseased cells from healthy cells such as, identifying cancer cells in a mixture of cells [39].

4.1. Introduction

The task of detecting interesting patterns is critical in many biomedical applications including ECG and MRI [102], [33], [34] in order to find useful insights towards disease diagnosis. With the advent of novel biological applications of micro and nanoscale devices [105], [36], [27], [37], we are able to detect the biological targets, such as DNA and human cells, at finer granularity. Unfortunately, the problem with these devices is that they the detection of biological targets with these devices have many challenges. The devices produce large amounts of raw data and the task of pattern detection is coupled with the enormity of the datasets [92], [38]. For example, the data collected from the translocation of a typical biopsy sample through a pore is about 10 GB. The state-of-the-art detection and analysis is based on the visual inspection, which is tedious, error-prone, and even an expert has to spend innumerable hours to analyze a blood sample, which is in the magnitude of gigabytes. In addition, the available software, such as pClamp can only analyze a subset of the total acquired data. Furthermore, due to the highly dynamic nature of data produced by the bio-nano sensors, the useful patterns are very sparse in the data and are orders of magnitude smaller than the total acquired data. In a clinical setup, the raw data collected from many

patients' biopsy samples, using multiple solid-state micropores quickly becomes too large to be handled by a single workstation. These challenges motivate the design of an automated and distributed detection technique, which can effectively acquire and process the raw data (collected from many micropores) for detecting and identifying useful patterns.

Recent trends show the efficacy of using distributed computing in genomics [106], [107], [108], [109], proteomics [110], [111], and other large-scale applications, such as physics [112] and astronomy [113].

In this paper, we design and develop a novel distributed technique that distributes/splits the data across multiple nodes and enables individual nodes to acquire and process its raw data segments to detect and identify useful pulses.

Furthermore, the integer processing and recursive moving-average filtering [92] and [114] makes the technique faster and amenable for online monitoring. To this end, our proposed framework performs the following steps:

- Splits the collected raw data into a number of segments, one for each participating node in the system.
- For an improved performance, each raw data segment is acquired by individual nodes using double buffers.
- Each node converts the raw data into integers for efficient processing, i.e., to avoid round-off errors and also reduce the total size of the acquired data.

The integer data is then smoothed using moving-average filtering to reduce the noise in the data. The useful patterns in the de-noised data are detected using a threshold that is based on the mean and standard-deviation of the data.

Evaluation of our system using the datasets of real cancer cells show that our technique can detect pulses with 63% accuracy, and process 10 gigabytes of raw data on an 8-node cluster in less than a minute, a task that would rather take several hours when using the extant manual process. Our framework

produces output in the form of scatter plots, which can be further used by physicians/scientists to infer useful information for disease diagnosis and useful decision-making.

The rest of the paper is organized as follows. In Section 4.2, we discuss the design and algorithm of our distributed framework. Section 4.3 presents our experimental setup and results with rigorous performance evaluation and comparison between the distributed system and single node implementation with an increasing size of input raw data and different size of double buffers. Finally, we discuss success stories and limitations in Section 4.4.

4.2. Design

We present the design and implementation of our framework for splitting and distributing the data stored on an NFS shared storage across multiple nodes. Each node then processes the raw data segment assigned to it in a SPMD (Single Program Multiple Data) style using the following software modules: pre-processor, smoother, detector, and post-processor. The high-level work flow of our distributed framework is shown in Fig 13. The pre-processor formats its segment of raw data and reads the data efficiently using double buffers. The smoother reduces the noise and eliminates baseline shifts.

The detector detects pulses in the data based on a threshold computed from the statistics of the smoothed data. Finally, the consolidated results from all the nodes are merged and delivered for further analysis.

Distributed Design:

The data is collected on the shared storage of an NFS server and then distributed among participating nodes such that each node gets its own data segment using the system modules as below. The pre-processor at each node retrieves its data segment from the shared NFS storage and optimizes the overall processing from two perspectives: (i) the pre-processor overlaps the data transfer with the computation using double buffering; and (ii) converts the raw data into an integer format to avoid round-off errors, as well as enable reliable and data efficient processing. The raw data segment is read from the storage device

in chunks for subsequent processing. However, the time required to process a given chunk comprises of the data transfer phase from the storage and the subsequent computation phase while the data is held in

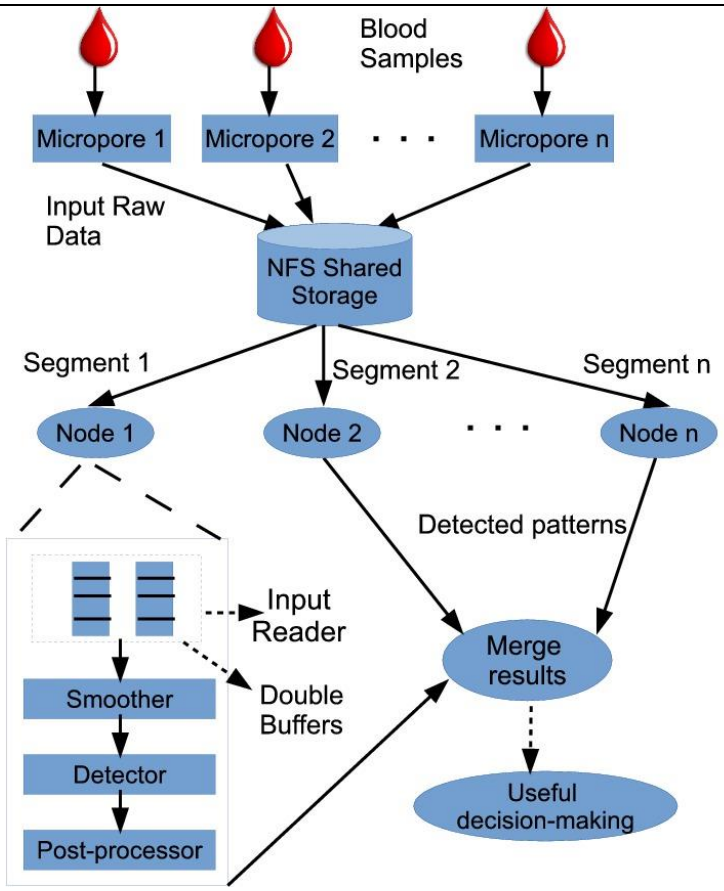


Fig 13. Distributed Detection Framework: Data can be collected from multiple bio-nano sensors on a shared storage of a high-speed server. Clusters nodes retrieve their own data segment from the shared storage and process it in parallel. Each data segment is processed by the individual node in four steps, including input reader, smoother, detector, and pre-processor that delivers result for useful decision-making.

the main memory of the node. Double buffering enables us to partition the main memory into two buffers, which are switched alternatively between the data transfer and the computation phase. For example, copying of the data chunk *k* in buffer *A* and the computation on chunk *k-1* in buffer *B* is achieved simultaneously. This enables processing of the data chunks in a pipeline fashion in order to improve the overall performance.

In the next step, the smoother removes noise and baseline wanders from the integer data. The cut-off frequency is an important factor used to decide band of frequencies that need to be passed in order to

only capture the interesting patterns in time domain. However, the cut-off frequency is inversely proportional to the size of the sampling window used in moving-average filtering. Higher cut-off frequency allows higher frequencies to pass, and thus results in slighter smoothing, and vice versa. In this case, the size of the sampling window is smaller. We tested the moving-average technique with different size of sampling window, i.e., 5, 10, and 20, and found 5-sample moving average to be the most effective for our problem.

Different techniques have been designed to detect peaks in temporal data [34]. The detector module detects pulses against a statistical threshold that is based on the mean and standard-deviation of the smoothed data. Furthermore, the mean and standard-deviation are computed from the number of samples equal to the size of window used in moving-average (5 in our case), and then used for the detection of patterns in the smoothed data samples as given by Eq. 19:

$$Threshold = mean - 4 \times std\text{-}deviation \quad (19)$$

The value of 4 in Eq. 19 is selected based on the empirical knowledge. However, our framework has the capability to automatically adapt to the changing characteristics (i.e., mean and standard-deviation) of the input data. The threshold is re-computed based on the initial k samples of the buffer of size N. The computed threshold stays constant for N-k data samples of a given buffer. In addition to that, our framework allows for experts to adjust the size of sampling window in moving average technique and the domain-specific value in Eq. 19 in order to further tailor and tune the detection according to the dynamic characteristics of the collected data.

In case of high variations in data, the buffer size can be reduced in order to allow the threshold to quickly adapt to the data and vice versa. The threshold detects the pulses along with their width and amplitude in smoothed data. The acceptable range of pulses for our problem is greater than 3 data samples, and less than 45 data samples. From the domain knowledge, we know that fewer samples (i.e., less than or equal to 3) are noisy pulses, while pulses with a width less than or equal to 45 data samples constitute a useful pulse that stems from a human cell [39]. The detected pulses along with their features

are delivered to the post-processor. The post-processor stores the features of the detected pulses in a comma separated values (csv) format on the storage device, which can be later used for further analysis in the form of scatter plots.

Algorithm

During the initialization phase, the data is split into number of segments equal to the number of nodes, and assigned to individual nodes according to static data distribution strategy. Each node then processes its data segment through several time-steps. Note that the size of double buffers is always less than or equal to the size of an individual data segment.

The duration of a time-step is directly proportional to the size of double buffers that is used. Larger buffers result in coarser time-steps and thus the given data segment can be processed within few time-steps. Conversely, smaller buffers result in fine-grained time-steps and therefore, requires large number of time-steps in order to process a given data segment.

Distributed Detection Algorithm:

Initialize();

For each sample in the data-segment

DataConversion();

Smoother();

Detector(); // detects patterns

For each of the detected patterns

ComputeFeatures();

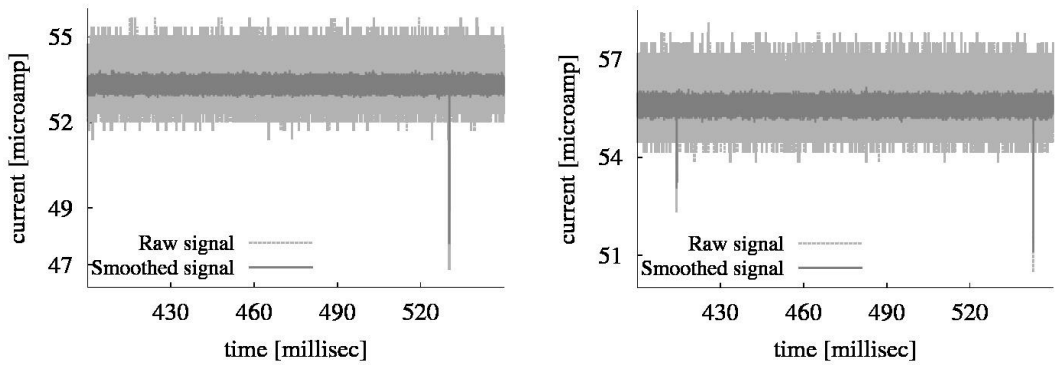
The algorithm shows that during a given time-step, each node concurrently processes its assigned data segment based on the offset, as shown in the following steps:

- Initially, each node reads the assigned data segment using double buffering technique and converts the raw data into integer data.
- The converted data is then de-noised using smoothing.

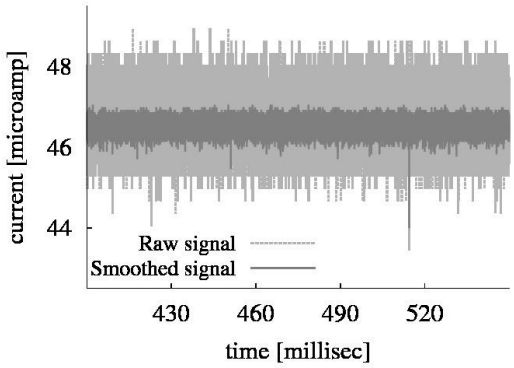
- Patterns in the de-noised data are detected against the threshold, which is computed from the statistics of the data.
- Compute useful features (i.e., width and amplitude) of the detected patterns.

4.3. Evaluation

In this section, we first explain the experimental setup of our target distributed system. Then, we show the impact of different levels of smoothing and threshold on the detection of different cell types and summarize statistics of distinguishing features of the detected pulses. Next, we analyze the speedup achieved on a distributed system in comparison to a single node for an increasing size of input data. We also examine the impact of the size of double buffers on overall execution time. Finally, we show the performance impact of using different interconnects for the nodes, such as 1 Gbps Ethernet and Infiniband.

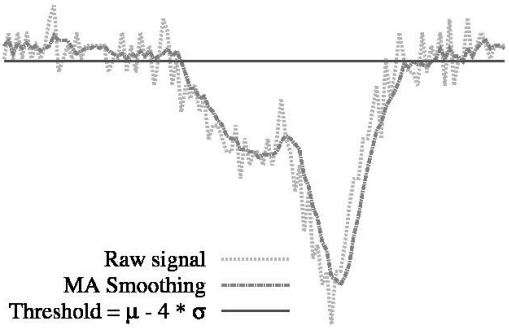


(a) Temporal data of Cancer Cells. (b) Temporal data of WBCs

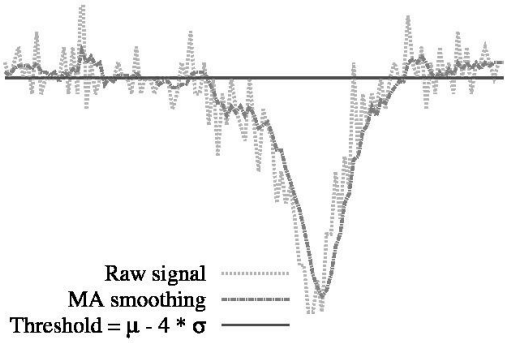


(c) Temporal data of RBCs

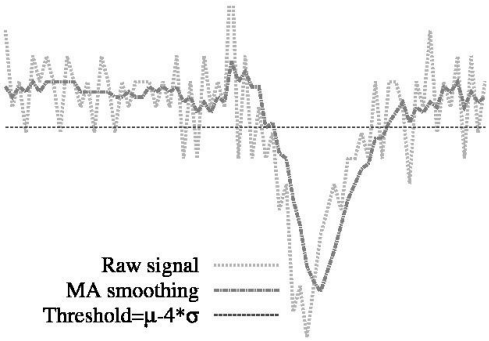
Fig 14. Noisy raw data and its smoothed version for different cell types with distinguishing pulses and the pulses in them are highlighted in insets. Different features of the RBCs, WBCs, and Cancer cells are summarized in Table 9. This demonstrates that cancer pulses are larger as compared to other cell types and retain their shapes even after smoothing.



(a) Typical Cancer pulse



(b) Typical WBC pulse



(c) Typical RBC pulse

Fig 15. Typical pulses from each cell type and their moving-average filtering with sampling window size of 5.

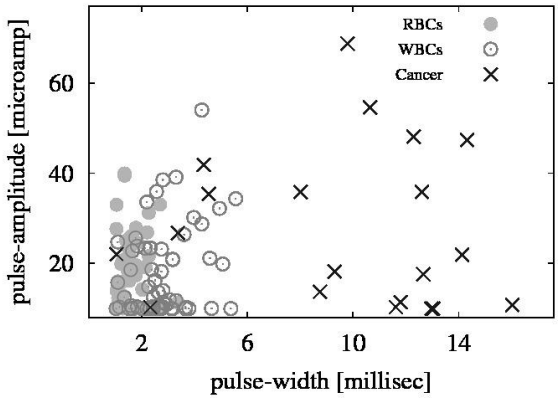


Fig 16. Scatter plot of different types of cells and their features detected from a mixture of cell types. Pulses detected in RBCs, WBCs and Cancer cells with Threshold = mean - 4 x std-deviation. The detected pulses are scattered instead of forming compact clusters because of the smoothing effect that results in reduced amplitude and increased width of the pulses.

Table 9. Summary statistics of pulse features from Red Blood Cells (RBCs), White Blood Cells (WBCs), and Cancer cells.

Pulse Features	RBCs	WBCs	Cancer Cells
Pulse Amplitude (microamp)	1.6 ± 5	3.14 ± 1	8.3 ± 4
Pulse Width (msec.)	15.0 ± 90	27.8 ± 31	30.1 ± 25

Table 10. The maximum size of double buffers that can be used in comparison to the integer data, i.e., converted from raw input data.

Raw Input Data			Integer Data	Double Buffers
Num. Profiles	Samples (Billion)	Size (GB)	Size (GB)	Max. Size (GB)
90	0.36	10	1.44	0.72
180	0.72	20	2.88	1.44
270	1.08	30	4.32	2.16
540	2.16	60	8.64	2.16

Table 11. Summary of the variation reduced in the raw data due to different levels of smoothing.

Sampling Type	RBCs	WBCs	Cancer Cells
Raw signal	±465.9	±469.1	±492.1
5-sample smoothing	±152.7	±188.9	±181.0
10-sample smoothing	±93.2	±143.3	±135.9
20-sample smoothing	±65.5	±124.8	±116.2

Datasets: The biological raw datasets are collected from the translocation of a typical biopsy sample via a micropore. The sample consists of Red Blood Cells (RBCs), White Blood Cells (WBCs), and Cancer cells. A typical profile of cells contains around 4 million samples recorded over a period of 2.2 microseconds. The overall data collected from a sample consists of 90 profiles (i.e., 360 million samples = 10 GB of raw data), as shown in Table **10**. Each sample has a resolution of 2 bytes and can measure up to 65,535 nanoamperes.

Micropore assembly: The biopsy sample is translocated through a Micropore of 12 micrometer radius and 200 nm long. The calibration of sampling frequency is an important factor to achieve the maximum throughput out of the pore. Decreasing sampling frequency results in a stable baseline with less noise, but cannot capture the useful translocation events at finer granularity. Conversely, higher sampling frequency results in the noisy data, which can suppress some of the translocation events. The optimal sampling frequency found for the micropore is 0.4 MHz.

A. Detection of Target Corpuscles

The level of threshold and the extent of smoothing are the two important parameters in order to effectively detect useful pulses. Changing the level of smoothing or threshold affects shape and count of the detected pulses, respectively.

Impact of Smoothing: Smoothing helps to eliminate the noise and baseline shifts, however, it also reduces information available in raw data, i.e., it affects shape of pulses, as shown in Fig 10. Larger smoothing reduces variations in the baseline to a greater extent, but at the cost of significantly changing the pulse shape. We achieved better smoothing of the pulse shape with a sample size of 5 as compared to a sample size of 10 and 20, which results in higher reduction of noise and deterioration of pulses. For clarity, we have only shown smoothing achieved by 5-sample moving average in Fig 11. The results also demonstrate that cancer pulses are larger than other cell types and retain their shapes even after smoothing, thus making them amenable to such proposed automated detection.

Impact of Threshold: Different thresholds result in different count of the detected pulses. Threshold closer to the baseline e.g., Threshold = mean - 3 *x* std-deviation results in the detection of noisy pulses (false positives) in addition to the useful pulses. However, threshold away from the baseline e.g., Threshold = mean - 5 *x* std-deviation results in miss-detection of useful pulses that are smaller in size (false negatives). Smoothing with 5-sample moving average followed by a Threshold = mean - 4 *x* std-deviation is found optimal for our datasets, as captured in Fig 11.

Detected Pulses:

Figure Fig **16** shows the scatter plots of the detected pulses from all the three different types of data. The plots show the width and amplitude of the detected pulses along x-axis and y-axis, respectively.

The reason for horizontal spread observed in the plot is due to smoothing that actually reduces the amplitude of the pulse by eliminating vertical fluctuations in the pulse, however, this increases the horizontal width of the pulses. We also observe overlap among different types of pulses. This is because that some of the smaller pulses in the cancer cluster closely resembled to the larger pulses of WBCs due to their similar widths and amplitudes. The average of the translocation time and the amplitude of the detected pulses show that these features are different for each cell type and that they differ significantly in their size and stiffness i.e., the extent to which they block the pore when the pass through it, as shown in Table 9.

Pulse Statistics: Increasing smoothing decreases variation in the data, as shown in Table 11. Greater smoothing (e.g., 20-sample moving-average) results in higher suppression of noise in the original signal (i.e., the standard deviation for RBCs is reduced from 465.9 nanoamperes to 65.5 nanoamperes), as compared to 10-sample and 5-sample moving average, which results in fairly small reduction in noise.

As show in Table 9, Cancer pulses have greater pulse amplitude and width than other cell types. Furthermore, the p-value < 0.01 for the translocation time shows that the cell types significantly differ from each other. In contrast, p-value > 0.01 for pulse amplitude shows that the cell types do not differ significantly.

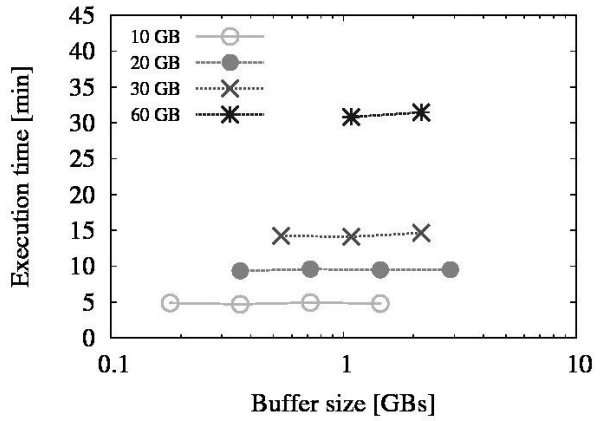


Fig 17. Scalability of our framework on a single node: Execution time for an increasing size of double buffers for an increasing input raw data (i.e., 10 GB, 20 GB, 30 GB, and 60 GB). Note that 10 GB corresponds to a single biopsy sample, 20 GB is equivalent to two biopsy samples, and so on.

B. Performance and Scalability

Next, we discuss the performance of automated pulse detection on a single node versus multiple nodes on a distributed system. We show the scalability achieved over multiple nodes. In addition, we also discuss the impact of changing the size of double buffers on overall execution time.

An Increasing Input Raw Data: The input data processed by our distributed framework ranges from 10 GB to 20 GB, 30 GB, and 60 GB. These datasets correspond to the raw data collected from typical blood sample(s) when translocated through a solid-state pore. Furthermore, when a 60 GB raw data is converted to integer data, it reduces to about 8.64 GB, slightly larger than the memory footprint of a cluster node, as shown in Table 10. The time taken to process these different datasets on a single node versus multiple distributed nodes is shown in Fig 17 and Fig 18, respectively.

Performance on a Single Node: We show the scalability achieved with a single node on our target cluster machine. The 10 GB raw data is processed for different size of double buffers (i.e., 180 MB, 360 MB, 720 MB, and 1.44 GB), 20 GB with a double buffer size of 360 MB, 720 MB, 1.44 GB, and 2.88 GB, and so on, as captured in Fig 17. The maximum size of double buffers is kept less than the total size of the integer data in order to make sure that we process the given data in chunks and do not overflow the

memory footprint of the machine. We observe linear scalability for an increasing size of input data. Additionally, we do not see significant change while changing the size of double buffers. However, we see slight degradation in performance, when we increase the size of double buffers i.e., in case of 30 GB and 60 GB.

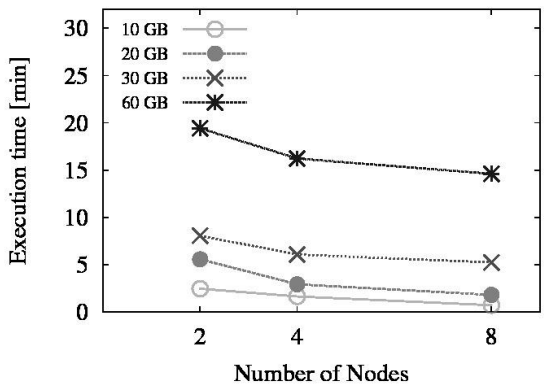


Fig 18. Framework scalability with an increasing size of double buffers. Scalability of our distributed framework on cluster nodes. Execution time for an increasing size of input raw data (i.e., 10 GB, 20 GB, 30 GB, and 60 GB). Double buffers of size 90MB is used to facilitate split/distribution of the given raw datasets across 8 nodes.

Impact of Double Buffers: The execution time with respect to different double buffer sizes for an increasing size of input raw data is shown in Fig 17. We do not see significant difference in the execution time for different double buffers. This shows that the results are scalable with varying size of double buffers. However, slight overhead is incurred in case of large buffer size, such as when using the 2.16 GB double buffers to process 60 GB of raw data, as captured in Fig 17.

Speedup Achieved on Multiple Nodes: Execution of our distributed detection system for 2, 4, and 8 nodes is shown in Fig 18. We observe the performance when the number of nodes is increased for a given dataset. However, we do not see significant performance improvement when the size of double buffers is larger, due to the overhead incurred in the critical path. Additionally, we are able to process 60 GB of raw data, collected from six biopsy samples, using 8 nodes in less than 15 minutes.

Overhead in Speedup: The overhead in speedup is smaller in the case of fewer nodes as compared to the large number of nodes as shown in Fig 19. In case of 2 nodes, the overhead in speedup is 5%, 16.2%,

14.2%, and 20.1% for an input data of 10 GB, 20 GB, 30 GB, and 60 GB, respectively. While in case of 4 nodes, we observe an overhead of 20%, 22%, 42%, and 53% for an input of 10 GB, 20 GB, 30 GB, and 60 GB, respectively. However, the parallelization of 10 GB, 20 GB, 30 GB, and 60 GB, results in 16%, 35.2%, 66%, and 78% overhead, respectively. We observe that the overhead increases with the increase in the size of input data.

Impact of Interconnects: We observe a large difference between the underlying interconnects, i.e., 1 Gbps Ethernet and Infiniband. The performance increase achieved from the Infiniband is 23.1% compared to 1 Gbps Ethernet, as shown in Fig 20.

Selection of Parameters: In case of moving-average filtering, a size of 5 for the sampling window is found optimal for the signal-to-noise ratio in our datasets. Based on the empirical knowledge, the value of $k=4$ is found suitable in order to subtract k times standard-deviation from the mean of the data and thus, compute the threshold, as shown in Eq. 19. Finally, the increase in size of double buffers is scalable with an increasing size of input data, as far as, their size (double buffers) does not overflow the memory footprint.

4.4. Discussion

In this work, we present a distributed detection approach that can acquire and process raw data collected from bio-nano sensors at an accelerated speed. In our experiments, we show that the designed framework can process data at a maximum throughput of 36 MB/sec. In other words, the framework has the ability to support the collection of raw data and its online processing from about 18 micropores simultaneously (each calibrated at a sampling rate of 2 MB/sec). In order to further accelerate the detection process, we need to incorporate fine-grained parallelism at the chunk level with the use of accelerator, such as, graphics processing units (GPUs). Such faster platforms will demand an increased sampling rate from the micropore-based experimental setup.

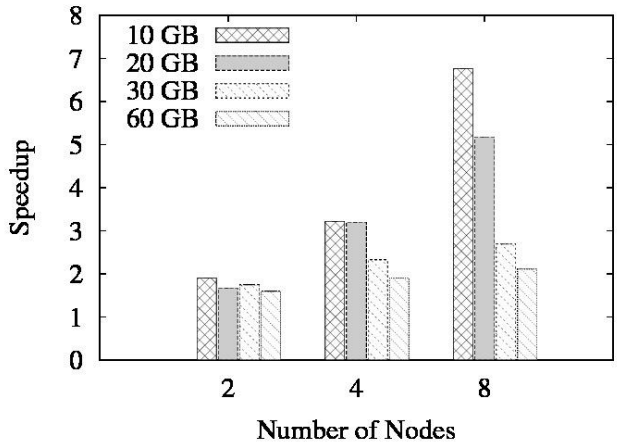


Fig 19. Speedup achieved by our distributed framework on cluster nodes for an increasing size of input raw data w.r.t. different number of nodes.

Nevertheless, the sampling rate depends on many factors including the speed and size of the biological targets that are translocated via micropores.

The accuracy achieved with our detection technique is about 63%, mainly because of the highly dynamic nature of data generated from bio-nano sensors, and secondly, due to the moving-average filtering and

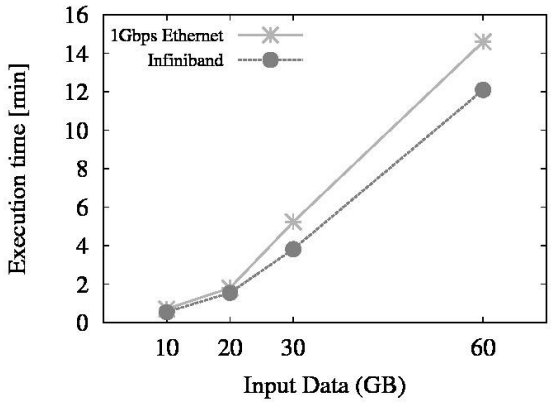


Fig 20. Comparison of Execution time on 1 Gbps Ethernet vs. Infiniband. Performance comparison of 1 Gbps Ethernet and Infiniband is tested on 8 nodes for an increasing size of input raw data (i.e., 10 GB, 20 GB, 30 GB, and 60 GB.

sophisticated approaches, such as machine learning techniques are needed. Such approaches, while computationally expensive, are more error-resilient and can differentiate between the noise and information (actual patterns) with an increased accuracy. While developing more robust and error-resilient detection algorithms, the need for an increased accuracy also stresses advancements on bio-nanotechnology to detect biological targets with an increased signal-to-noise ratio, while sensing them at finer granularity.

Our distributed computing framework splits the overall data on a shared storage among nodes, and enables each node to retrieve the assigned data chunk with its offset. Chances are rare for useful patterns to span across the boundaries of the split data segments and become a false alarm. The reason is that the patterns are very sparse in the data and the size of patterns is orders of magnitude smaller than a data segment (few bytes vs. gigabytes).

In addition to splitting the data into segments across nodes, we use double buffers within each node to overlap I/O with the computation, while enable processing of large datasets in chunks. Furthermore, to address large-scale processing of data, which is greater than the memory footprint, requires careful consideration of the memory usage. In order to achieve this, the breakdown for the size of double buffers is the physical memory limit. In our experiments, we are able to process 60 GB of raw data by keeping the aggregated size of double buffers to a maximum of 4.26 GB (2.13 GB for an individual buffer).

However, increase in the size of buffers results in an increase in the critical path of the overall computation, and become effective until after reading the very first chunk of a data segment. In addition to that, buffers larger than the size of memory footprint can either crash due to segmentation faults, or results in paging, if virtual memory is enabled. Conversely, very small buffers result in too much swapping and I/O that leads to an increased communication rather than the computation itself. Therefore, we need an optimal size of double buffers that is selected based on the total size of the input data in order to read large amount of raw data efficiently. Our experiments show that as far the size of double buffers is

less than the size of the integer version of input data and within the memory footprint, delivers an acceptable performance.

In our distributed setup, the splitting of data segments across nodes and the assignment of functional nodes is determined statically. In our future work, we aim to determine the status of nodes on fly and subsequently, utilize the node only if it is alive in order to achieve fault tolerance. Furthermore, we have assumed an even distribution of workload across the participating nodes.

This is possible in our framework, since the interesting patterns are very sparse in the datasets and therefore, a node will barely have larger number of patterns than its neighbors. Nonetheless, with the advent of high-performance sensors with high-quality of raw data (i.e., enhanced information with respect to the noise and baseline shifts), we will need to design dynamic load balancing techniques. One such strategy is to throttle the amount of data that is fed to a node on fly and assign future data to another node that is idle or relatively less burdened.

Chapter 5

Conclusion

In conclusion, bio-nano sensing approaches are promising to detect biological targets at finer granularity but are tied to collective challenges, including large amount of raw data and manual processing atop. This dissertation addresses these challenges by realizing software techniques – adapt to the noisy bio-nano devices and automate the detection of biological biomarkers – as well as expedite classification and identification of the diseased cells, etc.

Parallel implementation of detection techniques, while using GPUs; provide 5X faster system design as compared to the electrical measurement setup. In addition to that, a GPU-based approach coupled with advanced I/O techniques can detect biological molecules in the collected data in few minutes, which will otherwise take a couple of hours, if analyzed manually. We also provide results for static and dynamic detection techniques. Adaptive detection techniques are more robust to noise with an increased complexity and computation; however, the results are in a good agreement with the known biosets - within an error of 8.1%.

This work also addresses the challenge of detecting and classifying diseased cells in a blood sample that is actually a mixture of different types of human cells. Machine learning techniques including supervised and unsupervised algorithms have been adapted and trade-off between accuracy and performance has been discussed. *k*-Nearest Neighbor algorithm can classify different types of human cells to a higher accuracy at the cost of an increased percentage of training data; however, training part is computationally expensive. With the use of an 80% of training data, we were able to classify all types of human cells with 100% accuracy using useful features. In contrast, *k*-Means classifies only cancer cells with an accuracy of 100%; however, such an approach does not need training data – a computationally efficient approach. Nonetheless, these classifiers used the features of the pulses, which were detected with an accuracy of 70% in the typical biological assay taken from a blood sample.

Another important use case addressed by this work is to efficiently process large amounts of data for the detection of biological targets that can be collected from multiple bio-nano sensors simultaneously in an online clinical setup. Employing an 8-node cluster machine enables detection of cancer cells in a typical biological assay, mere 0.5 milliliter of blood, in less than a minute, as compared to manual detection, which will take a couple of hours and is also subject to personnel's experience. A comparison of results while varying number of nodes is investigated as well; however, increasing number of nodes does not show a linear speedup due to the I/O bottlenecks that mainly arise due to the shared storage device from which the data chunks are retrieved and split across the participating nodes. Nevertheless, such an investigation shows a promising result with distributed computing – leaving room for improved algorithms that can reduce execution stalls in a distributed setup.

5.1. Future Work

This work lays groundwork for automated disease diagnosis and innovative endeavors towards devising a statistical model for an accurate detection and classification of biological targets with the least number of tuning parameters and a high-throughput implementation. In addition to that, simulation of protein interactions which is crucial towards drug design and disease diagnosis is also discussed.

The ability to automatically adapt to the noisy data collected from bio-nano devices with varying noise levels and statistics, and subsequently, enable detection and classification of different types of biological targets, lies in a statistical model that will leverage advanced machine learning techniques, such as autoregressive moving-average, artificial neural networks, and support vector machines.

In a real clinical setup, where many patients are visiting and data is generated from different sensors including *solid-state micropores*, *bare micropores*, and *nanopores* with different data characteristics, such as varying mean and standard deviation, the detection objective becomes challenging. To cope with such a scenario, we need a model for threshold-based detection technique which can automatically adapt to such noisy data instead of tuning parameters, while hopping from one

sensor data to another. Filtering noise in the raw data followed by threshold modeled on statistics of the data will enable detection of targets with a higher accuracy. Consequently, we need classification algorithms to further classify the detected patterns. Our future work includes the design of combined model-based approach that can effectively filter and classify biological targets in the raw data with varying characteristics collected from different bio-nano sensors.

In a situation, where large amount of data is collected from many sensors, we need to accelerate the overall computations. The advanced I/O techniques coupled with GPU-based setup will accelerate the overall process by overlapping the computation with I/O [115]. Such framework has impact in a real clinical setup where many patients are visiting each day and the collected biological assay need to be analyzed for instantaneous feedback. Data collected from one blood sample easily results in few GBs of data. Thus, the data collected from hundreds of patients visiting per day will lead to TBs of data. The designed framework will enable to process the data in minutes, which otherwise consumes innumerable hours through manual processing. Additionally, the advanced I/O techniques will not only process data at higher sampling speed, but it will also ensure efficient utilization of the limited memory of the system by feeding it with the optimal chunk size. Furthermore, the useful patterns constitute very small fraction of the total acquired raw data. Recording only the required results and discarding the raw data will result in an efficient utilization of the attached storage to the system.

Such approaches will enable the detection of biological targets at finer granularity, an increased resolution, and a higher throughput. However, this leaves more room for the bio-nano measurement setup in terms of catch up. The bio-nano sensing measurement depends on many factors, such as length of the molecule, speed of the molecule, and the sampling rate of the sensor. The designed framework will only show the strength of the algorithms but will further help in accelerating the detection and classification of biological targets at large scale in order to support multiple sensors in an online setup.

Another important future direction that this work intrigues is the study of protein interactions that helps in understanding important protein behavior, such as folding and unfolding. However, the

simulation of proteins is highly compute-intensive and time-consuming, and the simulation of just an entire set of a bacterial cytoplasm takes almost a year to complete. Our goal is to explore the impact of large-scale distributed systems and cloud computing towards the acceleration of protein interactions. State-of-the-art protein simulations are limited to small number of biomolecules due to limited computing resources. Accelerating these simulations can help in understanding the unknown behavior of many proteins and thus, can help towards improved drug design which can ultimately help in disease diagnosis. However, uncovering useful insights about proteins interaction using *In vitro* experiments is a daunting task. The main bottleneck is that *In vivo* experiments limit the simulation of the whole set of biomolecules simultaneously. The reactions that occur in a living cell result into *In vivo* crowding. Single cell studies under cryoelectron tomography have revealed the crowded cytoplasm to *In vivo* models of cells and quantified the noise where the biomolecules follow non-uniform distribution [116-121]. Computational approaches exist that can theoretically predict the diffusion in the cytoplasm that stem from the crowding and related interactions [122-124]. Furthermore, how the crowding and diffusion of biomolecules affect the function of the cell is of important research. The designed simulations reduce the gap between *In vivo* and *In vitro* experiments by modeling the associated proteins at and up to the granularity of atomic level. The modeled rates of diffusion in proteins and respective thermodynamics stabilities almost match the *In vivo* results [124].

Whole the approach is promising, there are still some limitations. (1) The models simulate electrostatic potentials between macromolecules only, and not the hydrodynamic interactions (HI) between them due to limited computing resources. Part of reason is the inclusion of HI makes the diffusion coefficient (D_{trans}) slower as compared to the case that does not consider HI, and hence can further de-accelerate the simulation process. (2) The interaction of macromolecules requires modeling at micro- and nano-scale, which in turn necessitates very high resolution data collection – using time steps at picosecond intervals. Such resolution makes the simulations extremely hungry for the storage, memory, and computing power.

Computing power seems to be a bottleneck in enabling better discovery in this domain. For instance, the use of few GBs memory of a typical desktop machine restricted the types of macromolecules that can be modeled to mere 51 molecules, where many times more exist in the system. The data resolution similarly had to be reduced to enable an approximated computation.

In summary, the impact of a large-scale distributed computing in protein simulations:

1. The molecular trajectories of the atom dynamics, which are computed to simulate various behaviors of the proteins including folding, involve massive amounts of data. For instance, typical trajectories store at least 12 bytes of data per trajectory per atom, and a simulation can involve hundreds of millions of atoms, capturing the behavior at millions of time steps. This means a few GB are required per time-step [125] for bulky and complex macromolecular structures. State-of-the-art models are limited to simulate only a few microseconds of the interaction and used coarser time-steps. Even then using a desktop machine with few GBs of memory took several minutes to simulate one million time steps [126]. The entire simulation is expected to take almost a year [124]. Better system architectures, and the use of accelerators, such as GPUs, are promising here. The parallel implementation of trajectory simulations using GPUs will make the simulation faster. This will enable larger simulations in the same wall clock time as compared to the sequential simulation.
2. Typical cellular models consist of hundreds of millions of biomolecules, and simulation trajectories arising from such models, often stores thousands of frames. The storage requirements varies from hundreds of GBs to several TBs. Simulation of E. coli model [116] that consists of 20,000 active particles, while 600,000 obstacles and demands 500 simulations runs such that each run further consists of 5000 time steps. The storage requirement for such simulations demands up to 750 GB with 1.5 GB per simulation. Cloud-based technologies with huge amount of storage and memory can help in providing larger space to hold and process many times more simulations.

Such cloud-based setup will enable simulations of larger set of biomolecules. Furthermore, coupled with parallel implementation will further enable larger simulations achievable in the same wall-

clock time, as compared to non-cloud based setups. In general large-scale distributed systems or cloud computing will open endeavors for modeling protein interactions and other important intracellular environment. In a nutshell, this will help expediting custom drug design and related use cases.

Bibliography

[1] J. p. a. W. J. Tompkins, "A Real-Time QRS Detection Algorithm," *IEEE Transactions on Biomedical Engineering,* vol. 32, pp. 230-236, 1985.

[2] J. J. A. R. E. Barr, J. Sonnenfeld, "Peak-detection algorithm for EEG analysis," *Int J. Biomed. Comput, ,* vol. 9, 1978.

[3] I. N. S. C. Malpas, "Effect of asphyxia on the frequency and amplitude modulation of synchronized renal nerve activity in the cat," *J. Autonom. Nerv. Syst. ,* vol. 40, 1992.

[4] L. L. S. G. F. Dibona, S.Y. Jones, "Characteristics of renal sympathetic nerve activity in sodium-retaining disorders," *Am. J. Physiol.,* vol. 271, 1996.

[5] G. G. H. J.B. Korten, "Respiratory waveform pattern recognition using digital techniques," *Comput. Biol. Med,* vol. 19, 1989.

[6] D. L. K. R. J. Urban, E. Van Cauter, M.L. Johnson, J.D. Veldhuis, "Comparative assessments of objective peak-detection algorithms: II. Studies in men," *Am. J. Physiol.,* vol. 254, 1988.

[7] M. L. J. J.D. Veldhuis, "Cluster analysis: A simple, versatile, and robust algorithm for endocrine pulse detection," *Am. J. Physiol.,* vol. 250, 1986.

[8] J. J. V. W.J.E. De Ridder, J.A.C.A. Rombouts, J.M. Van Nueten, J.A.J. Schuurkes, "Computer-assisted method for the analysis of postprandial gastrointestinal motility in conscious dogs," *Med. Bio. Eng. Comput.,* vol. 27, 1989.

[9] A. H. L. R. A. Goubran, "Experimental signal analysis in ion mobility spectrometry," *Int J. Mass Spectron. Ion Proc. ,* vol. 104, 1991.

[10] K. R. Coombes, S. Tsavachidis, J. S. Morris, K. A. Baggerly, M. C. Hung, and H. M. Kuerer, "Improved peak detection and quantification of mass spectrometry data acquired from surface-enhanced laser desorption and ionization by denoising spectra with the undecimated discrete wavelet transform," *Proteomics,* vol. 5, pp. 4107-4117, Nov 2005.

[11] Z. H. C. Yang, and W. Yu,, "Comparison of public peak detection algorithms for MALDI mass spectrometry data analysis," *BMC Bioinformatics,* vol. 10, 2009.

[12] P. Du, Warren A. Kibbe, and Simon M. Lin, "Improved peak detection in mass spectrum by incorporating continuous wavelet transform-based pattern matching," *Bioinformatics,* vol. 22, pp. 2059-2065, 2006.

[13] R. A. e. al, "(The Milagro Collabortion). Evidence for TeV emission from GRB 970417a," *In Ap. J. Lett. ,* vol. 533, L119, 2000.

[14] A. J. Smith, "Milagro Collaboration: A search for bursts of tev gamma rays with milagro," *In Proceedings of the 27th International Cosmic Ray Conference (ICRC 2001), 07-15 August 2011, Hamburg, Germany,* 2001.

[15] H. S.-b. a. B. Shafai, "Detection of Peaks in Spectral Representation of Music Signals," *Communications in Computing,* 2004.

[16] R. B. Fisher and D. K. Naidu, "A Comparison of Algorithms for Subpixel Peak Detection " *Image Technology - Springer-Verlag, Heidelberg,* 1996.

[17] I. Azzini, R. Dell'Anna, F. C. F. Demichelis, A. Sboner, and E. B. A. Malossini, "Simple Methods for Peak Detection in Time Series Microarray Data," *Proc. CAMDA'04 (Critical Assessment of Microarray Data),* 2004.

[18] H. H. P. V. Hese, B. Vanrumste, Y. D'Asseler and P. Boon,, "Evaluation of Temporal and Spatial EEG Spike Detection Algorithms " *Fifth FTW PhD Symposium, Faculty of Engineering, Ghent University,* Dec 2004.

[19] Kleinberg, "Bursty and Hierarchical Structure in Streams," *In Proc of 8th ACM SIGKDD,* pp. 91 - 101, 2002.

[20] A. V. G. M. Ma, and P. Beukelman, "Developing and Implementing Peak Detection for Real-Time Registration " *In Proc. of the 16th Annual Workshop on Circuits, Systems & Signal Processing, ProRISC, 2005,* 2005.

[21] J. Li, M. Gershow, D. Stein, E. Brandin, and J. A. Golovchenko, "DNA molecules and configurations in a solid-state nanopore microscope," *Nature materials,* vol. 2, pp. 611-615, 2003.

[22] S. E. Cross, Y. S. Jin, J. Rao, and J. K. Gimzewski, "Nanomechanical analysis of cells from cancer patients," *Nature nanotechnology,* vol. 2, pp. 780-783, 2007.

[23] S. M. Iqbal, D. Akin, and R. Bashir, "Solid-state nanopore channels with DNA selectivity," *Nature nanotechnology,* vol. 2, pp. 243-248, 2007.

[24] J. Clarke, H. C. Wu, L. Jayasinghe, A. Patel, S. Reid, and H. Bayley, "Continuous base identification for single-molecule nanopore DNA sequencing," *Nature nanotechnology,* vol. 4, pp. 265-270, 2009.

[25] A. S. R. MacCarron*, H. Chang, E. S. Kolesar and S. M. Iqbal, "Examining the Current Charactersitics of Solid-State Micropores and Nanopores for Target Molecule Detection," *Biomedical Engineering Society Annual Fall Scientific meeting 2009,* 2009.

[26] M. I. Romero-Ortega, A. R. Butt, and S. M. Iqbal, "Carbon Nanotube Coated High-throughput Neurointerfaces in Assistive Environments," in *in Proc. of the 2nd International Conference on Pervasive Technologies Related to Assistive Environments (PETRA'09),* Corfu, Greece, 2009, pp. 6-13.

[27] D. Deamer, "Nanopore analysis of nucleic acids bound to exonucleases and polymerases," *Annual review of biophysics,* vol. 39, pp. 79-90, 2010.

[28] M. J. Kim YR, Lee IH, Kim S, Kim AG, Kim K, Namkoong K, Ko C: , "Nanopore sensor for fast label-free detection of short double-stranded DNAs," *Biosensors and Bioelectronics,* vol. 22, pp. 2926-2931, 2007.

[29] M. D. Lee SB, Trofin L, Nevanen TK, Soderlund H, Martin CR, "Antibody-based bio-nanotube membranes for enantiomeric drug separations," *Science,* vol. 296, 2002.

[30] S. M. Iqbal, D. Akin, and R. Bashir, "Solid-state nanopore channels with DNA selectivity," *Nature Nanotechnology 2,* pp. 243-248, 2007.

[31] S. M. Iqbal and R. Bashir, *Nanoelectronic-Based Detection for Biology and Medicine*: Springer Publishers, 2008.

[32] W. Asghar, A. Ilyas, J. Billo, and S. Iqbal, "Shrinking of Solid-state Nanopores by Direct Thermal Heating," *Nanoscale Research Letters,* vol. 6, p. 372, 2011.

[33] M. S. Lewicki, "A review of methods for spike sorting: the detection and classification of neural action potentials," *Network: Computation in Neural Systems,* vol. 9, pp. 53-78, 1998.

[34] G. K. Palshikar, "Simple Algorithms for Peak Detection in Time-Series," *in Proc. 1st Int. Conf. Advanced Data Analysis, Buisness Analytics and Intelligence,* 2009.

[35] S. C. a. S. M. I. M. M. Bellah, "Nanostructures for Medicine and Pharmaceuticals," *Journal of Nanomaterials,* 2012.

[36] D. H. Kim, N. Lu, R. Ma, Y. S. Kim, R. H. Kim, S. Wang, *et al.*, "Epidermal electronics," *science*, vol. 333, pp. 838-843, 2011.
[37] M. I. W. Asghar, A. S. Wadajkar, Y. Wan, A. Ilyas, K. T. Nguyen and S. M. Iqbal, "PLGA Micro and Nanoparticles Loaded into Gelatin Scaffold for Controlled Drug Release," *IEEE Transactions on Nanotechnology*, vol. 11, pp. 546-553, 2012.
[38] W. W. Haixun Wang, Jiong Yang, Philip S. Yu, "Clustering by pattern similarity in large data sets," *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pp. 394-405, 2002.
[39] W. Asghar, Y. Wan, A. Ilyas, R. Bachoo, Y. Kim, and S. M. Iqbal, "Electrical fingerprinting, 3D profiling and detection of tumor cells with solid-state micropores," *Lab on a Chip*, 2012.
[40] M. H. John D. Owens, David Luebke, Simon Green, John E. Stone and James C. Phillips, "GPU Computing," *Proc. IEEE* vol. 96, pp. 879-899, 2008.
[41] D. L. J. D. Owens, N. Govindaraju, M. Harris, J. Kruger, A.E. Lefohn, T.J. Purcell, "A survey of general-purpose computation on graphics hardware," *Comput. Graph. Forum*, vol. 26, pp. 80-113, 2007.
[42] I. GraphStream, "GraphStream scalable computing plateform (SCP)," 2006.
[43] R. Hussong, B. Gregorius, A. Tholey, and A. Hildebrandt, "Highly accelerated feature detection in proteomics data sets using modern graphics processing units," *Bioinformatics*, vol. 25, pp. 1937-1943, 2009.
[44] D. Sato, Y. Xie, J. N. Weiss, Z. Qu, A. Garfinkel, and A. R. Sanderson, "Acceleration of cardiac tissue simulation with graphic processing units," *Medical and Biological Engineering and Computing*, vol. 47, pp. 1011-1015, 2009.
[45] W. Shen, D. Wei, W. Xu, X. Zhu, and S. Yuan, "GPU-based parallelization for computer simulation of electrocardiogram," in *Ninth IEEE International Conference on Computer and Information Technology, CIT* 2009, pp. 280-284.
[46] Y. Zhuo, X. L. Wu, J. P. Haldar, T. Marin, and L. Z. P. Hwu Wm, "Using GPUs to accelerate advanced MRI reconstruction with field inhomogeneity compensation," *GPU computing gems, Emerald Edition. Elsevier*, 2011.
[47] A. Hafeez, W. Asghar, M. M. Rafique, S. M. Iqbal, and A. R. Butt, "GPU-based real-time detection and analysis of biological targets using solid-state nanopores," *Medical and Biological Engineering and Computing*, pp. 1-11, 2012.
[48] Y. Cao, D. Patnaik, S. Ponce, J. Archuleta, P. Butler, W. Feng, *et al.*, "Towards chip-on-chip neuroscience: fast mining of neuronal spike streams using graphics hardware," in *Proceedings of the 7th ACM international conference on Computing frontiers*, New York, NY, USA, 2010, pp. 1-10.
[49] M. M. Rafique, B. Rose, A. R. Butt, and D. S. Nikolopoulos, "CellMR: A framework for supporting mapreduce on asymmetric cell-based clusters," presented at the Proceedings of the 2009 IEEE International Symposium on Parallel\&Distributed Processing, 2009.
[50] M. M. Rafique, A. R. Butt, and D. S. Nikolopoulos, "Designing accelerator-based distributed systems for high performance," in *10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid)* 2010, pp. 165-174.
[51] B. S. T. a. D. C. Andrews, "The Identification of Peaks in Physiological Signals," *Computers and Biomedical Research*, vol. 32, pp. 322-335, 1999.
[52] S. A. V. Nijm G. M., Swiryn S., Larson A. C. , "Comparison of Signal Peak Detection Algorithm for Self-Gated Cardiac Cine MRI," *Computers in Cardiology*, 2007.

[53] G. C. Lange E., Reinert K., Kohlbacher O., Hildebrandt A.,, "High Accuracy Peak Picking of Proteomics Data using Wavelet Techniques," *in Proceeding of Pacific Symposium in Biocomputing, Maui, Hawaii, USA,* pp. 243-254, 2006.

[54] J. G. V. Ganti, and R. Ramakrishnan, "Demon: Data evolution and monitoring," *In Proceeding of the 16th International Conference on Data Engineering, San Deigo, California,* 2000.

[55] F. K. J. Gehrke, and D. Srivastava, "On computing correlated aggregates over continual data streams," *In Proc. ACM SIGMOD International Conf. on Managmenet of Data,* 2001.

[56] Y. K. A. C. Gilbert, S. Muthukrishnan, and M. Strauss, "Surfing wavelets on streams: One-pass summaries for approximate aggregate queries," *VLDB 2001, Morgan Kaufmann,* pp. 79-88, 2001.

[57] Y. Z. a. D. Shasha, "Statstream: Statistical monitoring of thousands of data streams in real time," *Proceedings of 28th International Conference on Very Large Data Bases, Hong Kong, China,* pp. 358-369, August 20-23, 2002.

[58] S. D. Zhu Y., "Efficient Elastic Burst Detection in Data Streams," *Proc. SIGKDD 2003 Conf,* pp. 336-345, 2003.

[59] K. Harmer, et al., "A peak-trough detection algorithm based on momentum," *CISP'08,* vol. 4, 2008.

[60] V. T. Jordanov, and Dave L. Hall, "Digital peak detector with noise threshold," *Nuclear Science Symposium Conference Record* vol. 1, 2002.

[61] J. C. Phillips, John E. Stone, and Klaus Schulten, "Adapting a message driven parallel application to GPU-accelerated clusters," *High Performance Computing, Networking, Storage and Analysis, 2008, SC 2008. International Conference for. IEEE, 2008,* 2008.

[62] Y. Tao, Hai Lin, and Hujun Bao, "GPU-based shooting and bouncing ray method for fast RCS prediction," *Antennas and Propagation, IEEE transactions,* vol. 58, pp. 494-502, 2010.

[63] S. S. Hampton, Sadaf R. Alam, Paul S. Crozier, and Pratul K. Agarwal, "Optimal utilization of heterogeneous resources for biomolecular simulations," *High Performance Computing, Networking, Storage and Analysis (SC), 2010 International Conference for. IEEE, 2010,* pp. 1-11, 2010.

[64] N. C, "NVIDIA CUDA Programming Guide," 2012.

[65] J. N. E. Lindholm, S. Oberman, J. Montrym, "NVIDIA Tesla: a unified graphics and computing architecture," *IEEE Micro,* vol. 28, pp. 39-55, 2008.

[66] NVIDIA, "Whitepaper NVIDIA Fermi G80," 2009.

[67] I. B. J. Nickolls, M. Garland, K. Skadron, "Scalable parallel programming with CUDA," *ACM Queue,* vol. 6, pp. 40-53, 2008.

[68] NVIDIA, "Whitepaper NVIDIA Kepler GK 110," 2012.

[69] J. E. Stone, David Gohara, and Guochun Shi, "OpenCL: A parallel programming standard for heterogeneous computing systems," *Computing in Science & Engineering,* vol. 12, 2010.

[70] S. L. G. M. Garland, J. Nickolls, J. Anderson, J. Hardwick, S. Morton, E. Phillips, Y. Zhang, V. Volkov, "Parallel computing experiences with CUDA," *IEEE Micro* vol. 28, pp. 2618-2640, 2008.

[71] J. W. D. V. Volkov, "Benchmarking GPUs to tune dense linear algebra," *Proceedings of the 2008 ACM/IEEE conference on Supercomputing, IEEE Press, Piscataway, NJ, USA,* pp. 1-11, 2008.

[72] T. Schiwietz, T. Chang, P. Speier, and R. Westermann, "MR image reconstruction using the GPU," in *Proceedings of SPIE*, 2006.

[73] D. H. J. Halder, S.-K. Song, and Z.-P. Liang, "Anatomically-constrained reconstruction from noisy data," *Magnetic Resonance in Imaging,* 2008.

[74] S. L. J. A. Fessler, V. T. Olafsson, H. R. Shi, and D. C. Noll "Toeplitz-based iterative image reconstruction for MRI with correction for magnetic field inhomogeneity " *IEEE Trans. Signal Process,* vol. 53 pp. 3393-3402, 2005

[75] M. W. K. P. Pruessmann, P. Bornert, and P. Boesiger "Advances in sensitivity encoding with arbitrary k-space trajectories " *Magn. Res. Med.* , vol. 46, pp. 638-651, 2001.

[76] D. C. N. B. P. Sutton, and J. A. Fessler. , "Fast iterative image reconstruction for MRI in the presence of field inhomogeneities " *IEEE Trans. Med. Imag.,* vol. 22, pp. 178-188, 2003.

[77] S. S. Stone, J. P. Haldar, S. C. Tsao, W. Hwu, B. P. Sutton, and Z. P. Liang, "Accelerating advanced MRI reconstructions on GPUs," *Journal of Parallel and Distributed Computing,* vol. 68, pp. 1307-1318, 2008.

[78] T. J. B. von Rymon-Lipinski and N. H. E. Keeve, "Fourier volume rendering on the GPU using a split-stream-FFT," in *Vision, Modeling, and Visualization* ed: Ios Pr Inc, 2004, p. 395.

[79] T. Sumanaweera and D. Liu, "Medical image reconstruction with the FFT," *GPU gems,* vol. 2, pp. 765-784, 2005.

[80] T. Y. Huang, Y. W. Tang, and S. Y. Ju, "Accelerating image registration of MRI by GPU-based parallel computation," *Magnetic Resonance Imaging,* vol. 9, pp. 712–716, 2011.

[81] S. Lim, K. Kwon, and B. S. Shin, "GPU-based interactive visualization framework for ultrasound datasets," *Computer Animation and Virtual Worlds,* vol. 20, pp. 11-23, 2009.

[82] J. Mathe, A. Aksimentiev, D. R. Nelson, K. Schulten, and A. Meller, "Orientation discrimination of single-stranded DNA inside the α-hemolysin membrane channel," *Proceedings of the National Academy of Sciences of the United States of America,* vol. 102, pp. 12377-12382, 2005.

[83] W. Asghar, A. Ilyas, R. R. Deshmukh, S. Sumitsawan, R. B. Timmons, and S. M. Iqbal, "Pulsed plasma polymerization for controlling shrinkage and surface composition of nanopores," *Nanotechnology,* vol. 22, p. 285304, 2011.

[84] B. M. Venkatesan, A. B. Shah, J. M. Zuo, and R. Bashir, "DNA Sensing Using Nanocrystalline Surface-Enhanced Al2O3 Nanopore Sensors," *Advanced Functional Materials,* vol. 20, pp. 1266-1275, 2010.

[85] J. B. H. Heng, C. Kim, T. Timp, R. Aksimentiev, A. Grinkova, Y. V. Sligar, S. Schulten, K. and Timp, G., "Sizing DNA using a nanometer-diameter pore," *Biophysical journal,* vol. 87, pp. 2905-2911, 2004.

[86] a. L. G.Y.H. Lee, C. T.,, "Biomechanics approaches to studying human diseases," *Trends Biotechnol.,,* vol. 25, pp. 111-118, 2007.

[87] F. M. M. Brandao, A., Barjas-Castro, M. L. et al.,, "Optical tweezers for measuring red blood cell elasticity: application to the study of drug response in sickle cell disease," *Eur. J. Haematol.,,* vol. 70, pp. 207-211, 2003.

[88] a. L. E. A. Evans, P. L. ,, "Intrinsic material properties of erythrocyte-membrane indicated by mechanical analysis of deformation," *Blood,* vol. 45, pp. 29-43, 1975.

[89] M. Lekka, P. Laidler, D. Gil, J. Lekki, Z. Stachura, and A. Z. Hrynkiewicz, "Elasticity of normal and cancerous human bladder cells studied by scanning force microscopy," *European Biophysics Journal,* vol. 28, pp. 312-316, 1999.

[90] A. S. G. Vona, M. Louha, V. Sitruk, S. Romana, K. Schutze, F. Capron, D. Franco, M. Pazzagli and M. Vekemanas, "Isolation by Size of Epithelial Tumor Cells," *Am. J. Phathol.,* vol. 156, 2000.

[91] M. G. O. L. Zabaglo, M. Parton A. Ring, I. E. Smith and M. Dowsett, "Cell filteration-laser scanning cytometry for the characterisation of circulating breast cancer cells," *Cytometry,* vol. 55, pp. 102-8, 2003.

[92] A. Hafeez, Asghar, W., Rafique, M. M., Iqbal, S. M., and Butt, A. R., "GPU-based real-time detection and analysis of biological targets using solid-state nanopores," *Medical and Biological Engineering and Computing,* pp. 1-11, 2012.

[93] A. Carbonaro and L. L. Sohn, "A resistive-pulse sensor chip for multianalyte immunoassays," *Lab on a Chip,* vol. 5, pp. 1155-1160, 2005.

[94] H. Chang, B. M. Venkatesan, S. M. Iqbal, G. Andreadakis, F. Kosari, G. Vasmatzis, *et al., DNA counterion current and saturation examined by a MEMS-based solid state nanopore sensor* vol. 8, 2006.

[95] M. S. Lewicki, "A review of methods for spike sorting: the detection and classification of neural action potentials," *Network-Computation in Neural Systems,* vol. 9, pp. R53-R78, Nov 1998.

[96] R. M. K. J.E. Hopcroft, "A n5/2 algorithm for maximum matchings in bipartite graphs," *SIAM J. Comput,* vol. 2, pp. 225-231, 1973.

[97] S. Few, "Data Visualization for Human Perception," *Encyclopedia of Human-Computer Interaction,* 2010.

[98] N. B. a. J. Hoberock, "A Productivity-Oriented Library for CUDA," *GPU Computing Gems, Jade Edition, W.-M. W. Hwu, Ed. Morgan-Kaufmann,* 2011.

[99] W. C. H. a. M. F. K. D. R. Engler, "C: A Language for High-Level, Efficient, and Machine-Independent Dynamic Code Generation," *Proc. 23rd Annual ACM Symp. Principles of Programming Languages,* 1996.

[100] B. Nichols, "Pthreads Programming: A POSIX Standard for Better Multiprocessing," *Reilly and Assoc.,* 1996.

[101] M. S. a. C. Chou, "A modified version of the K-means algorithm with a distance based on cluster symmetry," *IEEE Trans. Pattern Anal, Mach. Intell.,,* vol. vol. 23,, pp. pp.674-680, 2001.

[102] E. R. Jose Alvarez-Ramirez, Juan Carlos Echeverria, "Detrending fluctuation analysis based on moving average filtering," *Statistical Mechanics and its Applications,* vol. 354, pp. 199-219, 2005.

[103] C. A. a. J. Glasbey, R., "Fast computation of moving average and related filters in octagonal windows," *Pattern Recognition Lett,* vol. 18, pp. 555-566, 1997.

[104] S. W. Smith, "Digital Signal Processing: A Practical Guide For Engineers and Scientist," *Newnes,* pp. 343-350, 2003.

[105] S. M. Iqbal and R. Bashir, "Nanoelectronic-based detection for biology and medicine," in *Springer Handbook of Automation*, ed: Springer, 2009, pp. 1433-1449.
[106] K. Krampis, T. Booth, B. Chapman, B. Tiwari, M. Bicak, D. Field, *et al.*, "Cloud BioLinux: pre-configured and on-demand bioinformatics computing for the genomics community," *BMC bioinformatics*, vol. 13, p. 42, 2012.
[107] L. D. Stein, "The case for cloud computing in genome informatics," *Genome Biol*, vol. 11, p. 207, 2010.
[108] M. Baker, "Next-generation sequencing: adjusting to data overload," *nature methods*, vol. 7, pp. 495-499, 2010.
[109] R. C. Taylor, "An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics," *BMC bioinformatics*, vol. 11, p. S1, 2010.
[110] S. P. Brown and S. W. Muchmore, "High-throughput calculation of protein-ligand binding affinities: modification and adaptation of the MM-PBSA protocol to enterprise grid computing," *Journal of chemical information and modeling*, vol. 46, pp. 999-1005, 2006.
[111] J. Berger and M. Barkaoui, "A parallel hybrid genetic algorithm for the vehicle routing problem with time windows," *Computers & Operations Research*, vol. 31, pp. 2037-2053, 2004.
[112] E. Deelman, C. Kesselman, G. Mehta, L. Meshkat, L. Pearlman, K. Blackburn, *et al.*, "GriPhyN and LIGO, building a virtual data grid for gravitational wave scientists," in *High Performance Distributed Computing, 2002. HPDC-11 2002. Proceedings. 11th IEEE International Symposium on*, 2002, pp. 225-234.
[113] G. B. Berriman, E. Deelman, J. C. Good, J. C. Jacob, D. S. Katz, C. Kesselman, *et al.*, "Montage: a grid-enabled engine for delivering custom science-grade mosaics on demand," in *Astronomical Telescopes and Instrumentation*, 2004, pp. 221-232.
[114] C. A. Glasbey and R. Jones, "Fast computation of moving average and related filters in octagonal windows," *Pattern Recognition Letters*, vol. 18, pp. 555-565, 1997.
[115] A. R. B. M. M. Rafique, and D. S. Nikolopoulos, "A capabilities-aware framework for using computational accelerators in data-intensive computing," *Journal of Parallel and Distributed Computing*, vol. 71, pp. 185--197, 2011.
[116] A. M. E. Roberts, J. O. Ortiz, W. Baumeister, Z. Luthey-Schulten, *PLoS Computational Biology*, vol. 7, 2011.
[117] F. F. J. O. Ortiz, J. Kurner, A. A. Linaroudis, W. Baumeister, "Mapping 70S ribosomes in intact cells by cryoelectron tomography and pattern recognition," *J. Struct. Biol.*, vol. 156, 2006.
[118] B. J. B. L. R. Comolli, K. H. Downing, C. E. Siegerist, J. F. Banfield, "Three-dimensional analysis of the structure and ecology of a novel, ultra-small archaeon," *ISME Journal*, vol. 3, pp. 159-167, 2008.
[119] A. Briegel, D. R. Ortega, E. I. Tocheva, K. Wuichet, Z. Li, S. Chen, *et al.*, "Universal architecture of bacterial chemoreceptor arrays," *Proc Natl Acad Sci U S A*, vol. 106, pp. 17181-6, Oct 6 2009.
[120] M. Beck, J. A. Malmstrom, V. Lange, A. Schmidt, E. W. Deutsch, and R. Aebersold, "Visual proteomics of the human pathogen Leptospira interrogans," *Nat Methods*, vol. 6, pp. 817-23, Nov 2009.

[121] S. Kuhner, V. van Noort, M. J. Betts, A. Leo-Macias, C. Batisse, M. Rode, *et al.*, "Proteome organization in a genome-reduced bacterium," *Science*, vol. 326, pp. 1235-40, Nov 27 2009.
[122] D. S. Banks and C. Fradin, "Anomalous diffusion of proteins due to molecular crowding," *Biophys J*, vol. 89, pp. 2960-71, Nov 2005.
[123] D. Ridgway, G. Broderick, A. Lopez-Campistrous, M. Ru'aini, P. Winter, M. Hamilton, *et al.*, "Coarse-grained molecular simulation of diffusion and reaction kinetics in a crowded virtual cytoplasm," *Biophys J*, vol. 94, pp. 3748-59, May 15 2008.
[124] S. R. McGuffee and A. H. Elcock, "Diffusion, crowding & protein stability in a dynamic molecular model of the bacterial cytoplasm," *PLoS Comput Biol*, vol. 6, p. e1000694, Mar 2010.
[125] J. S. Michael Krone, Thomas Ertl, and Klaus Schulten, "Fast Visualization of Gaussian Density Surfaces for Molecular Dynamics and Particle System Trajectories " *In. Euro Association for Computer Graphics*, pp. 67-71, 2012.
[126] U. W. Tihamer Geyer, "An O(n2) approximation for hydrodynamic interactions in Brownian dynamics simulations," *J. of Chem Phys*, 2009.