computer methods and programs in biomedicine 134 (2016) 53–67

journal homepage: www.intl.elsevierhealth.com/journals/cmpb

An accelerated framework for the classification of biological targets from solid-state micropore data

Madiha Hanif [a,b,c], *Abdul Hafeez* [d], *Yusuf Suleman* [a,e], *M. Mustafa Rafique* [f], *Ali R. Butt* [d], *Samir M. Iqbal* [a,b,c,g,h,*]

[a] *Nano-Bio Lab, University of Texas at Arlington, Arlington, TX 76019*
[b] *Department of Bioengineering, University of Texas at Arlington, Arlington, TX 76019*
[c] *Nanotechnology Research Center, University of Texas at Arlington, Arlington, TX 76019*
[d] *Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, VA 24060*
[e] *Department of Computer Science and Engineering, University of Texas at Arlington, Arlington, TX 76019*
[f] *IBM Research, Ireland*
[g] *Department of Electrical Engineering, University of Texas at Arlington, Arlington, TX 76019*
[h] *Department of Urology, University of Texas Southwestern Medical Center at Dallas, Dallas, TX 75390, USA*

A R T I C L E I N F O

Article history:
Received 10 December 2015
Received in revised form
5 May 2016
Accepted 13 June 2016

Keywords:
Cancer detection
Pattern detection and classification
Run-time systems
Human cells
Solid-state micropores/nanopores

A B S T R A C T

Micro- and nanoscale systems have provided means to detect biological targets, such as DNA, proteins, and human cells, at ultrahigh sensitivity. However, these devices suffer from noise in the raw data, which continues to be significant as newer and devices that are more sensitive produce an increasing amount of data that needs to be analyzed. An important dimension that is often discounted in these systems is the ability to quickly process the measured data for an instant feedback. Realizing and developing algorithms for the accurate detection and classification of biological targets in realtime is vital. Toward this end, we describe a supervised machine-learning approach that records single cell events (pulses), computes useful pulse features, and classifies the future patterns into their respective types, such as cancerous/non-cancerous cells based on the training data. The approach detects cells with an accuracy of 70% from the raw data followed by an accurate classification when larger training sets are employed. The parallel implementation of the algorithm on graphics processing unit (GPU) demonstrates a speedup of three to four folds as compared to a serial implementation on an Intel Core i7 processor. This incredibly efficient GPU system is an effort to streamline the analysis of pulse data in an academic setting. This paper presents for the first time ever, a non-commercial technique using a GPU system for realtime analysis, paired with biological cluster targeting analysis.

* *Corresponding author.* Department of Electrical Engineering, Department of Bioengineering, University of Texas at Arlington, USA; Department of Urology, University of Texas Southwestern Medical Center at Dallas, USA. Tel.: +1 817 272 0228; fax: +1 817 272 7458.
E-mail address: smiqbal@uta.edu (S.M. Iqbal).

http://dx.doi.org/10.1016/j.cmpb.2016.06.001
0169-2607/© 2016 Elsevier Ireland Ltd. All rights reserved.

1. Introduction

Diseases such as cancer can be fully cured, if detected and treated at early stages. The traditional methods like magnetic resonance imaging (MRI) and cytology are intrusive and are not done as part of screening for cancer. The current methods cannot decode the type of cancer in a sample, such as liver cancer or a brain tumor. However, nanoscale biotech devices, for instance nanopores [1,2] and micropores [3,4] enable the translocation of biological targets, such as DNA and human cells in a biological assay at finer granularity. The Coulter counter was patented to detect small particles using resistive-pulse sensing with microscopic tubes, and as the analytes pass through the microchannels, dips in the electrical current are recorded [5–7]. This is the basis of electrical detection in micropores and nanopores. Nanopores provide the ability to detect one molecule at a time [8–17] and the ability to separate individual polymers [18–22]. However, nanometer-scale pores have previously had limitations including, fluctuation in pore currents [23,24], limited residence time of the molecule in the nanopore, nanopore clogging, and poor biomarker selectivity [25,26].

Various computational methods are used to analyze the data attained from nanometer-scale devices, and significant work is focused on the applications of supervised machine-learning algorithms [8,27–36] in order to classify important patterns in gene expression data [37–40]. Simple threshold based on peak-detection algorithms can detect useful patterns in the raw data emerging from ECG and mass spectroscopy [41–43]. A threshold is based on local minimum/maximum, mean, standard deviation, energy or entropy [44–46]. These strategies motivate the design of machine learning approaches for the effective detection and classification of biological targets in the raw data collected from bio-nano sensors.

Nanopores have applications such as rapid detection and characterization of molecules, while micropores are widely used for separating cells [47]. The sensors in this paper are minuscule channels made in thin silicon membranes. Their output is a current signal that is measured in micro- and nanoamperes. Research shows that cancer cells are softer and deform more readily than their healthy counterparts because of their elastic behavior [48]. Such behavior of diseased and healthy cells is recorded as varying patterns (pulses) in the output signal when passed through these devices [49]. The pulses occur at different scales and amplitudes due to the varying size and physical properties of human cells—stiffness and viscosity. Nevertheless, the data collected from such sensors suffer from a large amount of raw data riddled with sensor artifacts. In situ translocation of a characteristic biological assay (0.5 milliliter of a blood sample) results in 10 GB of raw data. The commercial software tools used for the analysis of raw data are limited to smaller datasets, and even a well-trained technician has to spend innumerable hours to process and analyze the data from a simple biological assay. The ability to address an increasing amount of raw data arising from bio-nano devices lies in machine-learning approaches for an effective detection and classification of biological targets in order to accomplish high-quality decision making.

Originally aimed for gaming, graphics processing units (GPUs) have evolved as accelerators into a gamut of compute-intensive scientific applications including bioinformatics and biomedical signal processing [50–68]. The GPU is what translates binary data from the central processing unit (CPU) and converts it into a picture. The tiny dots of an image displayed on a monitor are called pixels. The GPU decides how to use the pixels on a monitor to create an image to reflect the binary data sent to the system. These highly parallel architectures are becoming pervasive toward embarrassingly parallel applications due to their massively parallel architectures. GPUs host clustered cores called streaming processors (SPs), which are further grouped into streaming multiprocessors (SMs). There are varied memory spaces available that range from slower off-chip global memory to the faster on-chip shared memory. On-chip shared memory is faster than off-chip memory, but typically smaller. GPUs are good at doing many tasks at the same time. Programmers optimize memory accesses to global memory either by coalescing memory accesses to global memory or by exploiting the shared memory to reduce the off-chip memory overhead. In parallel, the k-nearest neighbor algorithm is used to analyze the data. This technique, in particular, is used because it is a simple analysis tool. However, it is also very effective in this experiment as it delivers results and the necessary classification required for these studies.

This paper describes a novel system-level design that detects pulses from the collected raw data of solid-state micropores, computes useful features of the detected pulses, and finally, classifies unknown samples based on the learned knowledge. The results can be readily used by physicians/scientists to infer useful information for disease diagnosis.

The approach to create a robust and real-time data analysis system included:

- First, a moving-average filtering technique was used to detect pulses from the raw data that stemmed from the diseased and healthy human cells.
- Important features including *width, amplitude, mean, standard deviation, falling slope* and *rising slope* of each pulse, and their statistical significance were computed.
- This enabled instant and reliable classification with the use of k-nearest neighbor technique based on the distinguishing features of the detected pulses.
- Finally, parallel k-nearest neighbor algorithm was implemented on GPU for biological target cluster detection in order to improve the overall performance of the system.

The implementation of the algorithms for detecting and classifying biological targets from the raw data is described first. The GPU architecture and programming model, as well as the design of the classification algorithms, system components, and experimental setup are discussed in Section 2. Section 3 elaborates the results. Finally, summary and future directions are concluded in Section 4.

2. Methods and experimental setup

2.1. *System overview*

The high-level detail of the system modules is provided in Fig. 1. The system is composed of several software compo-

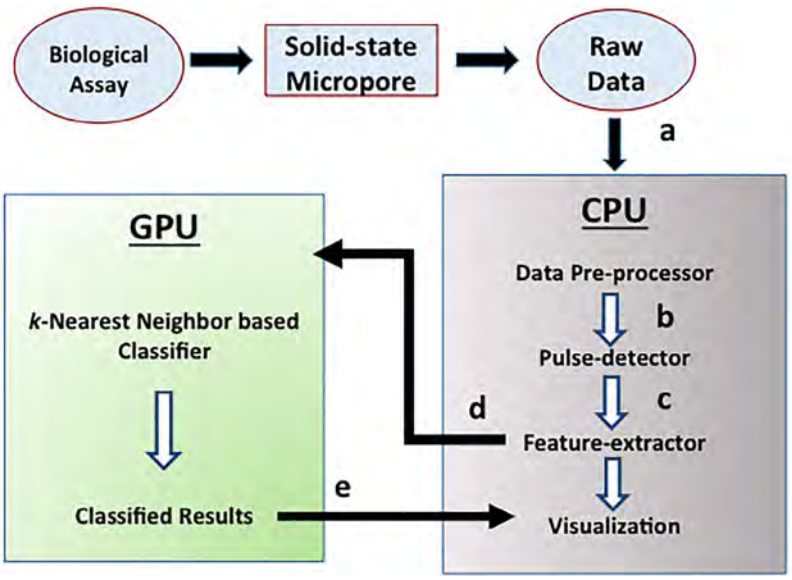

Fig. 1 – The system consists of five components (a–e). The data pre-processor fetches and formats the raw data, the pulse-detector detects pulses in them, and the feature-extractor computes features of the recorded pulse. The data are transferred to the GPU for parallel classification on the CUDA threads. GPU returns the results back to the CPU for visualization.

nents, including a data pre-processor, pulse-detector, feature-extractor, pulse-classifier, and visualization. The pre-processor component converts raw data into the appropriate format for efficient processing. Subsequently, the pulse-detector component detects pulses in the pre-processed data. The feature-extractor component computes important pulse features followed by the pulse-classifier that classifies the detected pulses into their respective groups based on their distinguishing features. Finally, the visualization module, which is extensively used in biomedical applications, produces 3D scatter plots.

2.2. *GPU architecture and programming model*

The programmers can harness the underlying massive parallel architecture of the GPUs through parallel programming tools. Nvidia has designed Compute Unified Device Architecture (CUDA) [69] that is used in most of the Nvidia GPUs, including Fermi [70] and Kepler [71,72]. The CUDA enables programmers to launch a compute-intensive kernel on GPU and harness a massive amount of computational power for the written programs to determine color, texture, and lighting of each pixel on the screen. Furthermore, CUDA provides abstractions to implement parallelism from coarser to finer granularity, in addition to the task parallelism. CUDA follows Single Program Multiple Data (SPMD) programming style. This enables programmers to launch compute- and data-parallel kernels onto GPU. CUDA thread is the smallest unit of computation on a typical GPU. To run very efficiently on a GPU, there needs to be many hundreds of threads; the more threads, the better. The CUDA is well known for its impressive number crunching and handling of large datasets. A kernel is primarily composed of a grid of CUDA threads that is organized into warps, which are further combined into blocks and altogether the blocks form a grid. The threads would all be executing the same function but with different datasets. The execution configuration for a given thread launch ranges from 64 to 512 threads per block. Threads within a block have access to the common shared memory, which is otherwise not shared across blocks. If needed, threads within a block can also perform barrier synchronization [50].

2.3. *GPU-based machine-learning algorithm (k-nearest neighbor)*

This algorithm is primarily organized into three steps. Initially, the Euclidean distance between each test sample to every training sample is calculated, which yields an n x m matrix, where n is the total number of test samples, and m is the entire number of training samples. Next, all the training samples are sorted with respect to each test sample in order to achieve a row-wise sorted matrix. Finally, the test sample is assigned using a class-based approach on the maximum proportion in a set of k selected training samples [73]. These steps are shown in Fig. 2, separated by *synchronization* barriers that guarantee the completion of previous steps before proceeding to the next step. In the following, we present mathematical formulation of the aforementioned steps.

2.4. *Mathematical formula*

The mathematical calculation of the aforementioned steps is done as follows:

2.4.1. *Euclidean distance*

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

$$D_{ij} = D_{ji} = \sqrt{\left[(i_1 - j_1)^2 + (i_2 - j_2)^2 + \ldots + (i_n - j_n)^2\right]} \qquad (1)$$

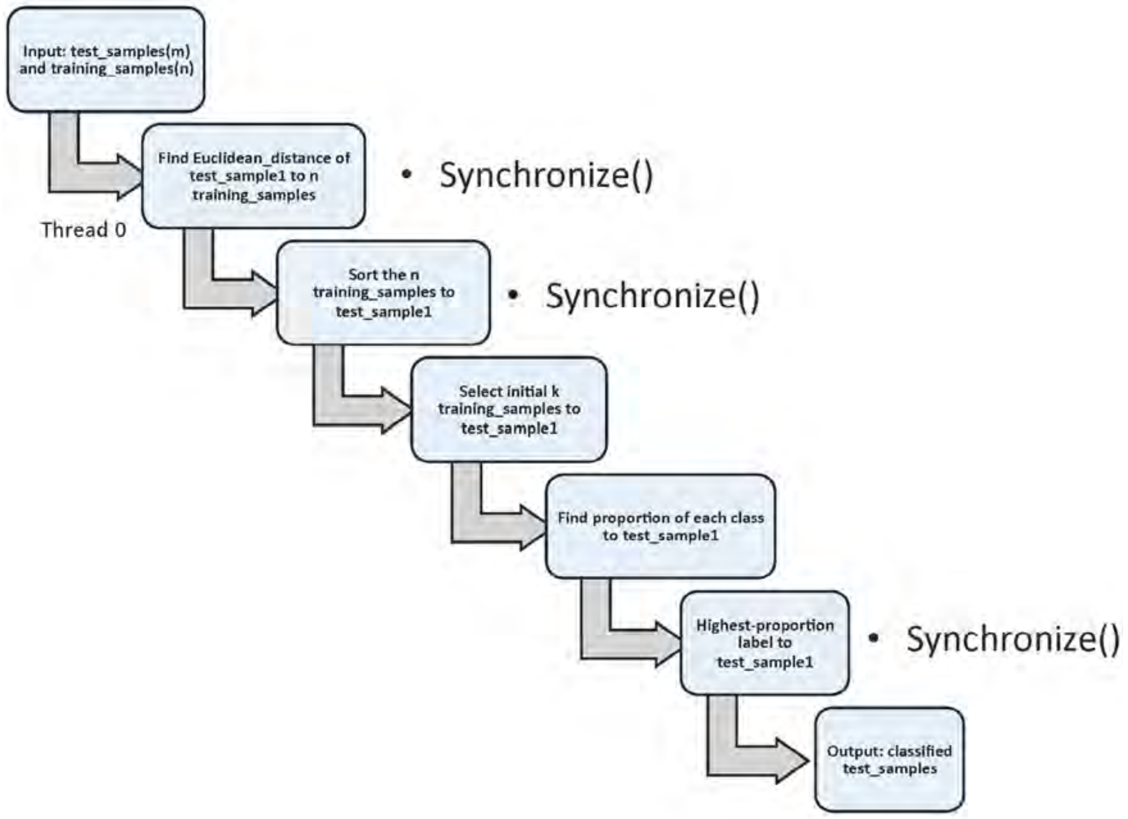
Input: test_samples(m) and training_samples(n)
Find Euclidean_distance of test_sample1 to n training_samples
• Synchronize()
Thread 0
Sort the n training_samples to test_sample1
• Synchronize()
Select initial k training_samples to test_sample1
Find proportion of each class to test_sample1
Highest-proportion label to test_sample1
• Synchronize()
Output: classified test_samples

Fig. 2 – Block diagram of *k*-nearest neighbor algorithm. The first step finds the Euclidean distance from all test samples to the training samples, and then it is synchronized. 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, and it is synchronized again. Finally, *k* training samples are selected from each set and based on the maximum proportion of class; the test sample is then synchronized and classified to that class. This algorithm is repeated for threads 1, 2 and so on.

Here, *i* and *j* are the indices of the samples in matrix A and B that corresponds to *test* and *training* sets, respectively. The goal was to compute distance from each sample in test (vertex) to every sample in the training (edge) set, i.e., *test (0, 1. . . n)*, and *training (0, 1. . . m)*. This evolved into a fully connected *bipartite graph* [41]. The algorithmic complexity was $O(|A| * |B|)$, where $|A|$ and $|B|$ were the cardinality of matrix A and B, i.e., *n* and *m*, respectively, and the running time was $O(m \times n)$.

To achieve the task of distance computation in parallel, the workload was distributed among threads such that each thread calculated one of the $n \times m$ distances. The indices per *test* and *training* samples were computed as: $test_{id} = thread_{id}/m$, and $training_{id} = thread_{id}\%m$ where $thread_{id} = (0, 1, \ldots, n \times m)$ in order to make sure that the distance from each test sample to every training sample was computed—employing 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*$_j$) was given by:

$$D_{ij} = \sqrt{\begin{bmatrix}(test_i \cdot width - training_j \cdot width)^2 \\ + (test_i \cdot amplitude - training_j \cdot amplitude)^2 \\ + (test_i \cdot hybrid - training_j \cdot hybrid)^2\end{bmatrix}} \quad (2)$$

2.4.2. *Sorting distance matrix*

Once the distances were computed from each test sample to every training sample, an *n* x *m* distance matrix was achieved, such that d_{ij} in the distance matrix was the distance between i^{th} test sample to the j^{th} training sample as shown below:

$$D_{ij} = \begin{bmatrix} d_{11}' & d_{12}' & \cdots & d_{1m}' \\ d_{21}'' & d_{22}'' & \cdots & d_{2m}'' \\ \cdots & \cdots & \cdots & \cdots \\ d_{n1}''' & d_{n2}''' & \cdots & d_{nm}''' \end{bmatrix} \quad (3)$$

Each row was 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.

$$D_{ij} = \begin{bmatrix} d_{11} & d_{12} & \cdots & d_{1m} \\ d_{21} & d_{22} & \cdots & d_{2m} \\ \cdots & \cdots & \cdots & \cdots \\ d_{n1} & d_{n2} & \cdots & d_{nm} \end{bmatrix} \quad (4)$$

2.4.3. *Decision in* k *neighborhood*

Finally, first *k* training samples were selected for each test sample in order to reduce the search space, and to classify the test sample based on the maximum participation of training samples from a particular class.

$$D_{ij} = \begin{bmatrix} d_{11} & d_{12} & \cdots & d_{1k} \\ d_{21} & d_{22} & \cdots & d_{2k} \\ \cdots & \cdots & \cdots & \cdots \\ d_{n1} & d_{n2} & \cdots & d_{nk} \end{bmatrix} \tag{5}$$

$$P_{ij} = \begin{bmatrix} prop_{11} & prop_{12} & \cdots & prop_{1c} \\ prop_{21} & prop_{22} & \cdots & prop_{2c} \\ \cdots & \cdots & \cdots & \cdots \\ prop_{n1} & prop_{n2} & \cdots & prop_{nc} \end{bmatrix} \tag{6}$$

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

2.5. *System components*

The system comprised of a data pre-processor to format the raw data; the pulse-detector to detect pulses in the formatted data; the feature-extractor to compute useful features of the detected pulses followed by the pulse-classifier that classified the pulses based on their features; and finally, the visualization module to analyze the results as scatter plots.

2.5.1. *Data pre-processor*

The acquired raw data was composed of current values in microamperes sampled at a few microseconds apart with a micropore-based electrical measurement setup. The data pre-processor efficiently read data from the storage device into the system. The acquired raw data was further converted into integers instead of floating-point values for efficient processing and to avoid round-off errors. It was then ready for the pulse-detector as shown with labels (a) and (b) in Fig. 1.

Double buffering [74] was used in the system to overlap the reading latency from the slower storage device with the computation [68]. Such configuration used two buffers and two threads, that is, the producer and consumer. The producer fetched data from the storage, while the consumer processed the data in the buffer in parallel. While the producer received data chunk *k+1* into one buffer at time *t+1*, the consumer was processing data chunk *k* in another buffer at time *t+1* in which the producer at time *t* received the data. This resulted in a sampling speed 1.6 times faster than the naive implementation where the data were first collected into the main memory from the storage device, and then computed. The data were transferred to the pulse-detector module for the detection of cells.

2.5.2. *Pulse-detector*

This module detected the translocation patterns of red blood cells (RBCs), white blood cells (WBCs), and cancer cells in the form of pulses passed through a micropore, as shown in Fig. 1(c). The technique used by the detector was based on the moving-average technique. This technique worked on the pre-processed integer data and avoided pre-processing steps like smoothing, and thus, eliminated subjectivity to some extent. The size of sampling window was critical to use the moving-average filtering. Increasing the size of the sampling window increased smoothing of the target data; however, it missed the useful pulses (false negatives) that were smaller. On the other hand, decreasing the size of the sampling window resulted in limited smoothing but detected noisy pulses as useful pulses (false positives). Averages of the sampling window were computed further to compare subsequent data samples to it. In case the next data sample was smaller than the computed average, the sample was recorded as candidate pulse. Finally, the subsequent data samples that were less than the threshold were recorded only as a pulse if the number of these samples were within the acceptable range as explained previously [38].

Different threshold values resulted in different number of detected cell types. If the threshold was kept too close to the baseline, large numbers of cells were detected. In contrast, if the threshold was away from the baseline, fewer cells of each type were detected. Smaller pulses constituting noisy pulses (false positives) were detected in addition to the actual pulses when threshold was closer to the baseline. Unfortunately, the number of false positives was large in the case of RBCs due to their smaller dimensions—closely resembling to the actual pulses, which made it challenging to discriminate these from the real pulses. However, in case of cancer cells, due to their large dimensions, the discrimination from noisy pulses was significant and therefore very convenient for the threshold detector. Conversely, when the threshold shifted away from the baseline, no noisy pulses were detected; however, it missed useful pulses (false-negative) due to their smaller amplitude. False negatives were frequent in the case of RBCs due to smaller sized pulses, and rare in the case of cancer cells due to their significantly larger dimensions. The detected pulses from different cell types were delivered to the feature extractor module to compute pulse features.

2.5.3. *Feature extractor*

This module took input from the pulse-detector as shown in Fig. 1(c). Input included the detected pulses in the data. Feature-extractor computed the important features of the detected pulses, such as width, amplitude, slopes, and statistics, which were then used by the next module on GPU for further pattern classification as shown in Fig. 1(d). The pulse width and amplitude alone were 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 were computed. *Statistical Feature*, such as the average of mean and standard deviation of each pulse were also computed to facilitate visualization in 3D.

When the current signal fell below the given threshold of baseline current and reverted to its original level, all the current values ($p_1, p_2 \ldots p_n$) were included in this dip to form a pulse. The duration between the start of the pulse and end of the pulse was the pulse width ($t_k - t_1$), where $t_1, t_2 \ldots t_k$ were the corresponding time instances at which the pulse current values were sampled. However, the pulse amplitude was the minimum value among the recorded values of the pulse, i.e., $p_{\min} = \min_{i=1} k\{p_i\}$. *Rising Slope* was 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* measured how abruptly the pulse fell toward its minimum and was defined as $m_{fall} = \frac{p_{\min} - p_i}{t_{\min} - t_i}$. The case for the numerator becoming zero (very rare) was also taken care of, so that the system did not fall into a halted state. Consequently, the system did not record such event as a pulse. The *Mean* was simply the average of the recorded values per pulse, $\mu = \frac{\sum_{i=1}^{k} p_i}{k}$, and the *Standard Deviation* per pulse was given by $s = \frac{\sqrt{\sum_{i=1}^{k} (p_i - \mu)^2}}{k}$. The *Hybrid Feature* of a pulse was computed as the average of mean and standard deviation, i.e., $\frac{\mu + s}{2}$.

The above features were further used not only to visualize the natural clusters of different cell types, but also to separate these out with the least amount of overlap in them. The rising and falling edges parameters were the least useful feature. However, pulse width, amplitude, and mean parameters were the best features to discriminate the clusters using 3D scatter plots, in addition to the width, amplitude and standard deviation. Usefulness was determined by the differentiation between WBCs and cancer cells. Unfortunately, the scatter plots still resulted in 50% overlap between WBC clusters and cancer clusters. The ultimate goal was to minimize this overlap as much as possible. However, with little more visual inspection *width, amplitude,* and *statistical feature* were found to completely remove the overlap between WBC and cancer clusters.

2.5.4. *Pulse-classifier*

Once the features for each pulse were computed, these were fed to the pulse classifier to classify the pulses into their respective clusters. The pulse classifier used *k*-nearest neighbor machine-learning technique to detect cancer clusters and to deliver results to the visualization module as shown in Fig. 1(e).

The *k*-nearest neighbor grouped the overall data into the training and testing samples, and based on the training samples, the test samples were classified. The pulse-classifier could be executed either on the CPU or on the GPU, as selected by the user. The GPU used multiple threads in order to run machine-learning algorithm in parallel, as explained below. The test samples (*n*) were classified using *k*-nearest neighbor classification technique based on different percentages of training samples (*m*). Generally, a larger proportion of training samples resulted in better classification of the test samples.

To compute each Euclidean distance in parallel, threads equal to $n \times m$ were launched. However, the amount of training samples and the remaining test samples impacted the total amount of threads to compute the Euclidean distance. That is, the total number of threads launched could be at least $n \times 1$, or $1 \times m$ when the proportion of training samples was huge ($n >> 1$) and vice versa. The maximum number of threads that could be deployed was $n \times m$, when 50% of data was used as training samples. The 50% proportion required approximately 130x131 = 17161 threads in our case, accomplished by 67 blocks of 256 threads each. The output of the Euclidean-distance kernel was the distance-matrix that contained test samples as rows, and training samples as columns. This distance-matrix was fed to the sorting kernel in order to sort the matrix row-wise, i.e., to achieve sorted training samples in ascending order per test sample based on its distance from that test sample. Finally, the sorted distance-matrix was input to the decision-making kernel. This kernel selected the first *k* sorted training samples ($m_1, m_2, \ldots, m_k$) for each test sample and computed the proportions of each class ($prop_1, prop_2, \ldots, prop_k$) in them. The maximum proportion out of these was calculated such that: $prop_{\max} = \max_{i=1} k\{prop_i\}$, and the test sample was then assigned to the class *i* that had the largest proportion in its *k* neighborhood.

2.5.5. *Visualization*

Visualization presents statistical information effectively in abstract form [75]. The results were organized into 3D scatter plots for pattern analysis of the biological targets. By visual inspection, *width, amplitude, mean*, and *standard deviation* were found the best for discriminating the clusters of RBC, WBC, and cancer. However, 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*. Therefore, mean and standard deviation were combined as *a statistical feature* to be used as a third dimensional feature to the width and amplitude, and was found to completely separate out the cancer cells from WBCs and RBCs.

2.6. *Experimental setup*

The experimental setup consisted of an Intel Core i7 for a GPU and CPU setup. CUDA, pthreads, and written code were involved in analyzing the data that were attained by running the biological assays through the solid-state micropore.

2.6.1. *Core i7-based GPU setup*

The experimental setup consisted of an Intel Core i7-based compute node with NVIDIA Quadro FX 580 and 6 GB of main memory installed on it. The CPU had 4 cores clocked at 1.6 GHz. The GPU has 512 MB global memory, 64 KB constant memory, 16 KB per block shared memory and 8K registers. Moreover, the GPU contained 32 cores, i.e., 4 multiprocessors (MP), with 8 cores per MP, clocked at 1.12 GHz. The system used 64 bit Linux Ubuntu 10.04.4 LTS, kernel version 2.6.32.

2.6.2. *CUDA*

With the given hardware and CUDA capability 1.1, the execution configuration could have a maximum of 512 threads per block. In order to achieve sorting required by *k*-nearest neighbor technique, we used CUDA SDK sorting. Quick sort implementation was used for CPU-based counter execution [76]. Built-in CUDA timer and event functions, i.e., *cudaEventCreate()*, *cudaEventRecord()* and *cudaEventSynchronize()* were used to measure the execution of kernel functions within the machine-learning algorithm. These routines ensured that previously issued CUDA events were recorded on the GPU. Finally, CUDA *syncthreads()* was used for synchronization across all thread blocks.

2.6.3. *Pthreads*

Pthread library was employed to implement *double buffering*. During our experiments, buffers of size 200,000 samples were found optimal for our problem [77]. The integer size on 64 bit Intel machine running Linux OS with gcc complier was 4 bytes. The time and current values in integers per sample contributed to 8 bytes per sample, and overall results were limited to 1.6 MB per buffer.

2.6.4. *Lines of code*

The system was implemented using C and CUDA with 349 lines of code for moving-average filtering, 53 for feature extraction, and 600 for *k*-nearest neighbor algorithm.

2.6.5. *Datasets*

The system was evaluated using three different biological assays describing several profiles of RBCs, WBCs and cancer cells. The data were collected from the micropore experimental setup with a pore diameter of 12 micrometers [78]. These bio-sets consisted of current values in microamperes sampled at a rate of 2.2 microseconds. The raw data were around 10 GB owing to 360 million current values. After pre-fetching the data into main memory and pre-processing the raw data into integers, the total amount of data shrank down to 2.88 GB, resulting in 68% reduction of the total acquired raw data. The resulting data could easily fit in the available 6 GB main memory and therefore avoided the system to page.

2.6.6. *Solid-state micropores*

The micropores are an example of bioMEMS devices that can detect human cells at the finer granularity from a biological assay (blood samples) and provide efficient, cheaper, and highly sensitive alternatives. The pore used for experiments had 12-micrometer radius and was made in 200 nanometer thin membrane that translocated single human cell at a given time. In order to achieve maximum throughput, the flow rate was optimized. Increasing flow rate (such as 20 microliter per minute) missed some of the translocation events, while decreasing it to 5 microliter per minute reduced the throughput significantly. An optimal throughput was achieved at 10 microliter per minute. In addition to that, sampling frequency of the micropore was an important factor to determine the desired output. Higher sampling rates induced lot of noise in the output current from the micropore and suppressed many translocation events, while lower sampling rates resulted in a more stable baseline current with a higher signal to noise ratio, but the system missed faster translocation events mainly caused by smaller sized cells, such as RBCs. An optimized sampling rate was achieved at 2.5 microseconds.

3. Results

3.1. *Detection accuracy*

Empirically, pulses with a width larger than four samples (approximately nine microseconds) and amplitude larger than 1000 nanoamperes were recorded in the evaluation. In this implementation, the moving-average filtering used a sampling window of 2000 samples to achieve a better detection of pulses. This reduced the pre-processed data to a total of 261 pulses—26 RBCs, 208 WBCs, and 27 cancer cells. Furthermore, we detected cancer cells with an accuracy of 70%.

The standard deviations (noise) of the acquired data from cancer cells, WBCs and RBCs were found to be 492, 499, 463 nanoamps, respectively. Therefore, cancer cells and WBCs required the threshold away from the baseline, as compared to RBCs. The threshold was thus set to 0.036 to detect cancer cells and WBCs. However, a threshold of 0.026 was chosen for the precise detection of RBCs, as shown in Fig. 3. The baseline shift and the follow up of moving average filtering are also shown in Fig. 3(a). The onsets of the raw data attained are displayed in Fig. 4(a), (b), and (c) which capture finer details of each remarkable pulse. Clearly, cancer cell pulses were larger when compared to WBCs and RBCs. RBC pulses were fluctuating and unstable as compared to the cancer and WBC pulses. The two edges emerging out in the RBC pulses led to an insight that RBCs were clumped together when passed through the micropore—most probably due to their smaller size [78].

Additionally, the features of the detected pulses such as pulse-width, pulse-amplitude, mean, standard deviation, falling slope, and rising slope were computed from the recorded samples per pulse. The width, amplitude, mean, standard deviation, falling slope, and rising slope are graphically defined in Fig. 3(d). Pulse-width, pulse-amplitude, and the statistical feature (combined mean and standard deviation) were found statistically significant features in separating out the clusters, and therefore, were used for further analysis of different cluster types. The next section shows the statistical significance of the distinguishing features.

3.2. *Statistical significance of features*

It was found that the average pulse amplitude of the cancer cells was larger than that of WBCs and RBCs as shown in Table 1. A similar trend was also observed for the average translocation time, and statistical feature. However, in the case of falling slope, it was observed that WBCs' average was greater when compared to the cancer cells, and similarly, in the case of rising slope, a greater average of RBCs was observed than

Table 1 – The average of the features for three types of pulses.

Cell types	Average pulse amplitude (microamp)	Average translocation time (microseconds)	Average of statistical feature (microamp)	Average of falling slope feature	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

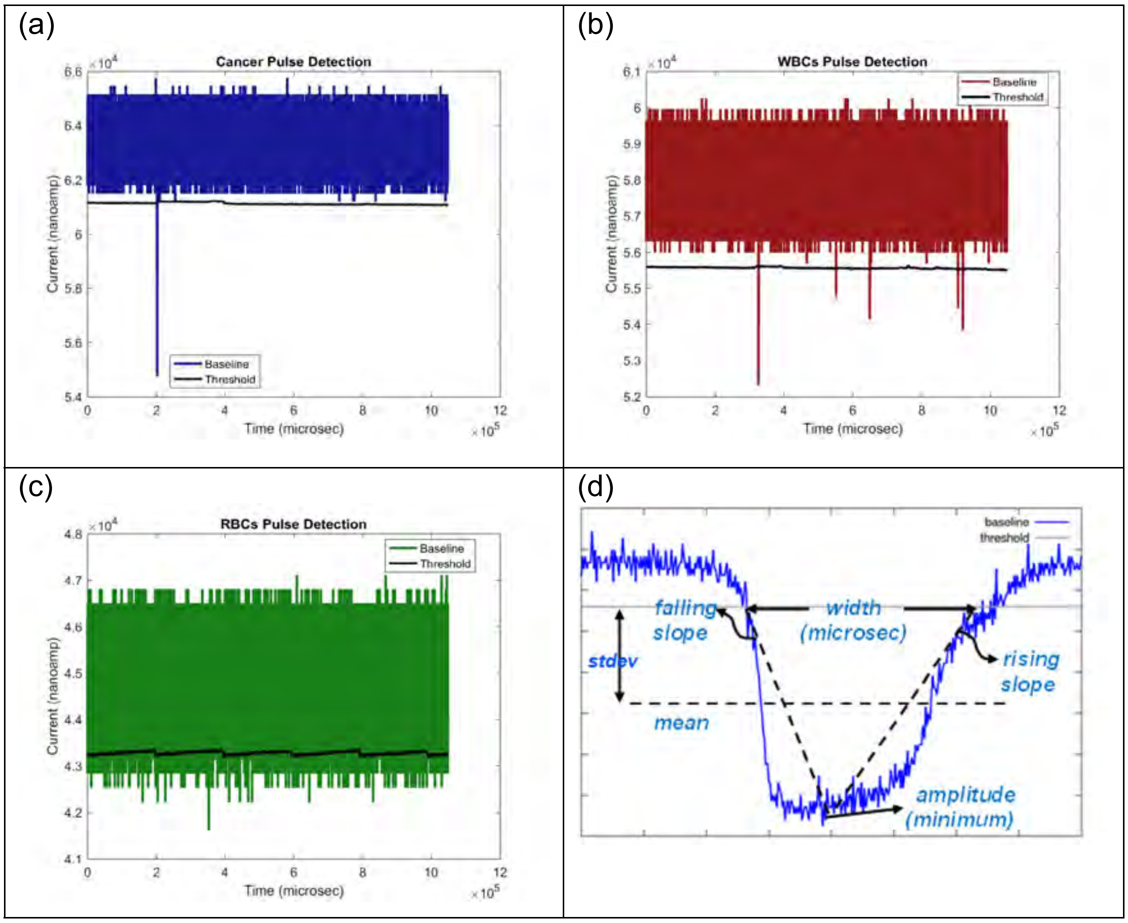

Fig. 3 – Detection results with the black line as moving-average based threshold. Raw data from micropore for (a) cancer cell translocation with baseline shift, (b) WBC translocation, (c) RBC translocation, (d) Typical pulse showing width, amplitude (minimum), falling slope and rising slope.

that of cancer cells. Such behavior of slopes, although useful for biophysical insights, did not make them useful for distinguishing the three types of human cells.

3.3. *ANOVA of pulse features*

A single-factor analysis of variance (ANOVA) was done for the pulse translocation time (pulse-width), pulse-amplitude, falling slope, rising slope and statistical features. We found p-value < 0.001 for pulse-width, pulse-amplitude, and statistical feature, and therefore, these were significantly distinguishing features as compared to the slopes whose p-value > 0.05. Furthermore, the F-test was done on the features. 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 the smaller values of F, which means scattered clusters resulting in overlapped regions. The features are given below in the order of their significance with the highest F-value for statistical feature:

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

i.e., $3290.6 > 179.9 > 129.5 > 2.9 > 0.38$

3.4. *Classification results*

The k-nearest neighbor is a supervised machine learning technique, where the data are first trained 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 of the data including 20%, 50% and 80% were used as training data, and k was kept 5 and 10, as shown in Figs. 5, 6 and 7. Accurate results were observed for larger amounts of training data. Furthermore, since the cancer cluster was already separated out from the other two clusters using a statistical feature, which otherwise overlapped, the cancer cluster was detected with 100% accuracy in all cases except for 20% training data and when $k = 10$, as explained in the following section.

3.4.1. *Case with 20% training sample data and* $k = 5$ *vs.* $k = 10$

In this case, since the size of the training data was small, there was not enough information to classify the test data accurately. Given total samples of 261; 20% of RBCs, WBCs and cancer cells corresponded to 5 RBCs, 42 WBCs, and 5 cancer cells respectively.

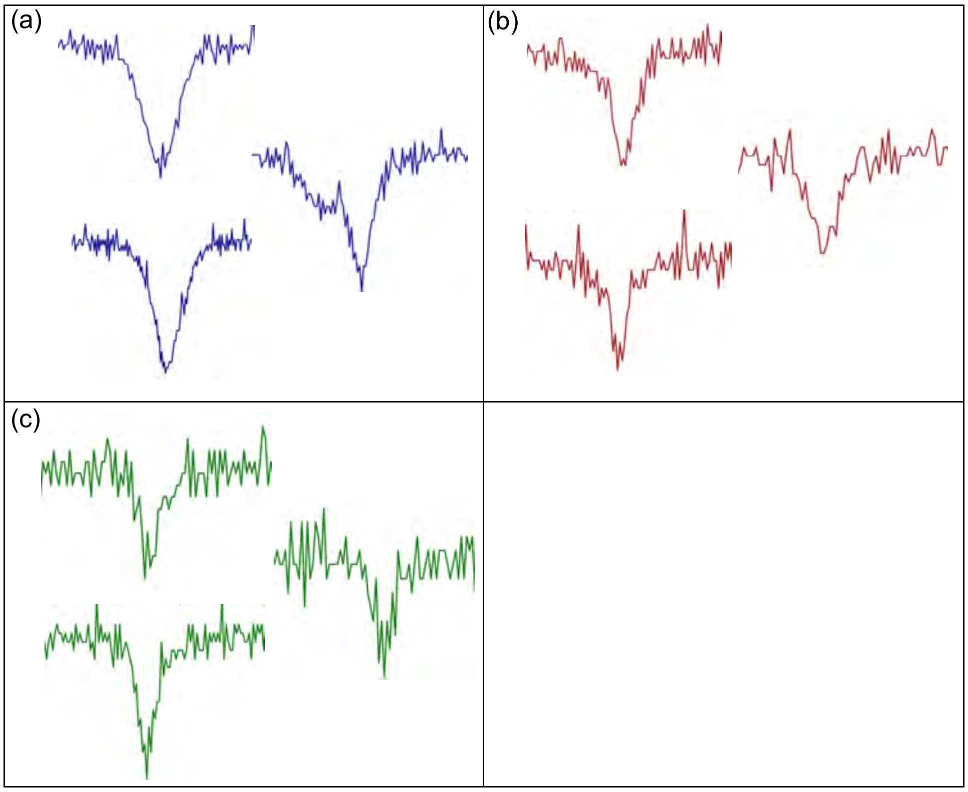

Fig. 4 – The onsets of the detection results with the black line as the moving-average based threshold. Three instances for each category of cells are represented (not to scale): (a) cancer cells, (b) WBCs, and (c) RBCs.

In the case of $k = 10$, only one cancer cell was classified incorrectly as WBC, as shown in Fig. 5. However, all RBCs were classified as WBCs, again due to the small size of training data, which did not capture enough information to allow RBCs to be classified accurately. Another reason for this misclassification was the large neighborhood as compared to $k = 5$ (Fig. 5), 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 a large proportion of WBCs, in case of $k = 10$ points was also responsible for the misclassification of one cancer cell.

3.4.2. *Case with 50% training sample data and* $k = 5$ *vs.* $k = 10$

In this case, the cancer cluster is classified accurately. However, half of the input data (detected pulses) were used as training samples, while the other half were used as test data (Fig. 6).

For $k = 5$, all RBCs were classified correctly as shown in Fig. 6(a). However, 25 WBCs were misclassified as RBCs. So there were 25 *false-positives* in the RBC cluster, while there were 25 *false-negatives* in the WBC cluster. In case of $k = 10$, 18 WBCs were misclassified as RBCs as shown in Fig. 6(b). This showed that 50% of the training data did not have enough representation of WBCs to classify these correctly. Furthermore, the significant overlap between RBCs and WBCs resulted in such misclassification.

3.4.3. *Case for 80% training sample data and* $k = 5$ *vs.* $k = 10$

Increasing the size of the training data reduced the misclassification rate of WBCs. Only 2 WBCs were detected incorrectly as RBCs in case of $k = 5$ as shown in Fig. 7(a). However, in the case of $k = 10$, it was able to classify the remaining 20% test data accurately for all types of cells (Fig. 7(b)).

3.5. *Accuracy of classification*

The accuracy measured using a contingency table for 50% and 80% of training data is shown in Table 2. 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, 208 were WBCs, and 27 were cancer cells. With 80% of training samples from each cell type, the 20% of the test data were actually 6 RBCs, 42 WBCs, and 6 cancer cells. However, 2 false-positives resulted in the case of RBCs—stemming from the misclassification of WBCs as RBCs. Furthermore, in the case of 50% training samples, the number of false-negatives in WBCs was 25 and 18, for a neighborhood size (k) of 5, and 10, respectively, which were miss-classified as RBCs.

3.6. *Performance results*

We now present the performance of the classification algorithms on both CPU and GPU. The Euclidean distance

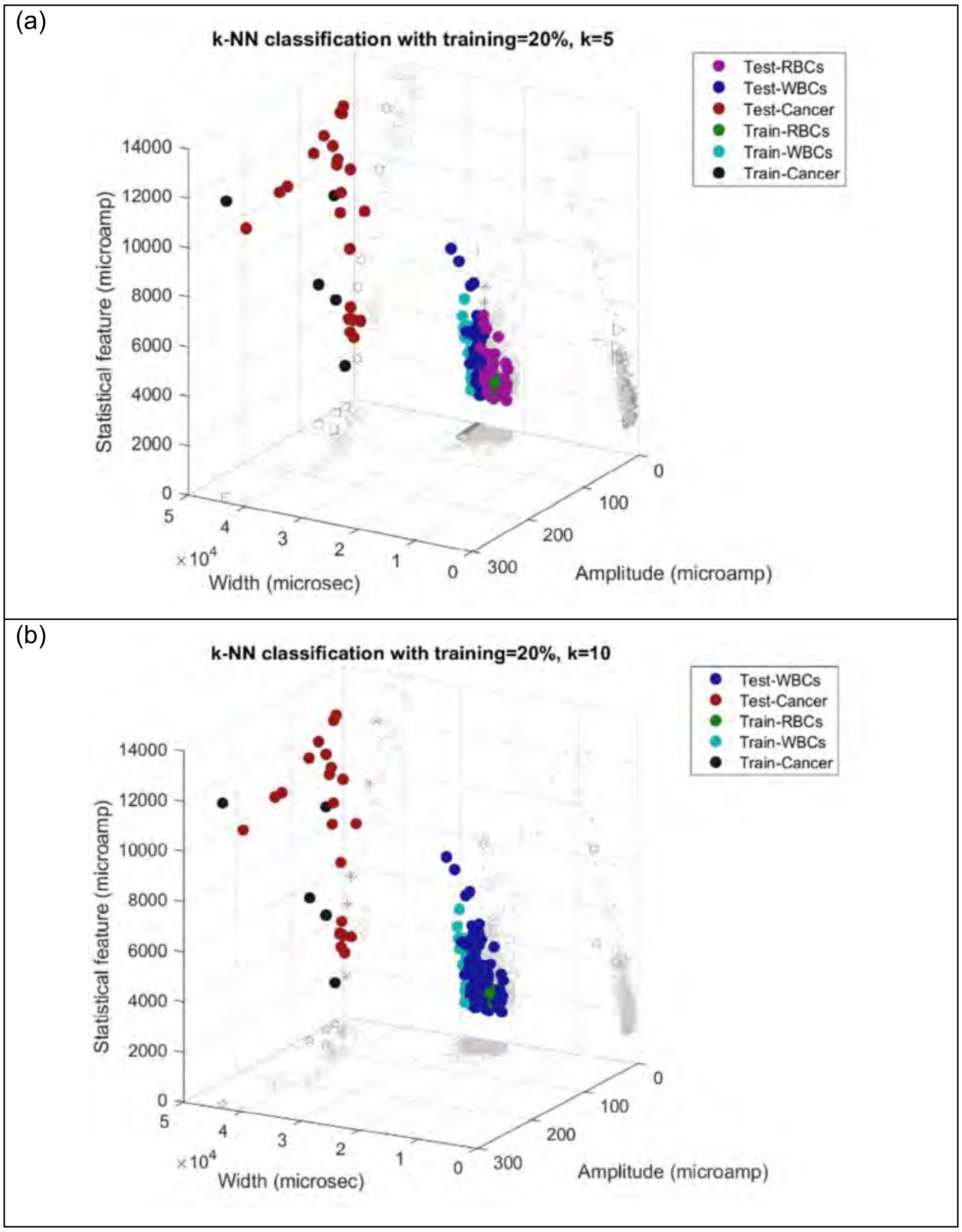

Fig. 5 – The results of *k*-nearest neighbor algorithm with training sample of 20% and (a) *k* = 5 and (b) *k* = 10.

computation resulted in computations three times faster on the GPU than the CPU (Table 3). The sorting was also almost four times faster on the GPU. The sorting algorithm was based on the CUDA SDK sorting networks, which used a bitonic sorter to construct a network using a parallel algorithm. The input data to the sorting algorithm was considered a batch and was divided into fixed size arrays such that the array-length was of size 2^n. In our case, the array-length and the batch-size referred to the number of the training, and test samples, respectively. In order to tailor the data accordingly, 256 data samples were used instead of 261, by removing 5 WBCs, while not changing the amount of RBCs and cancer cells. Reducing the number of WBCs, however, did not affect the classification of cancer cells. The sorting algorithm was tested with 25%, 50%, and 75% of training data that pertained to 64, 128, and 192 of array-length, respectively. In case of 75%, training data were increased to 256 samples from 192 samples by padding zeros in order to make it an integral power of 2. The data were sorted in ascending order after the padded zeros, which were grouped together at the beginning of the sorted array. The decision-making step was then implemented only on the CPU due to branch divergence, which resulted in performance degradation on the GPU.

4. Discussion

The speedup of 3–4X achieved from machine learning algorithm on GPU was fascinating. However, the experiments were

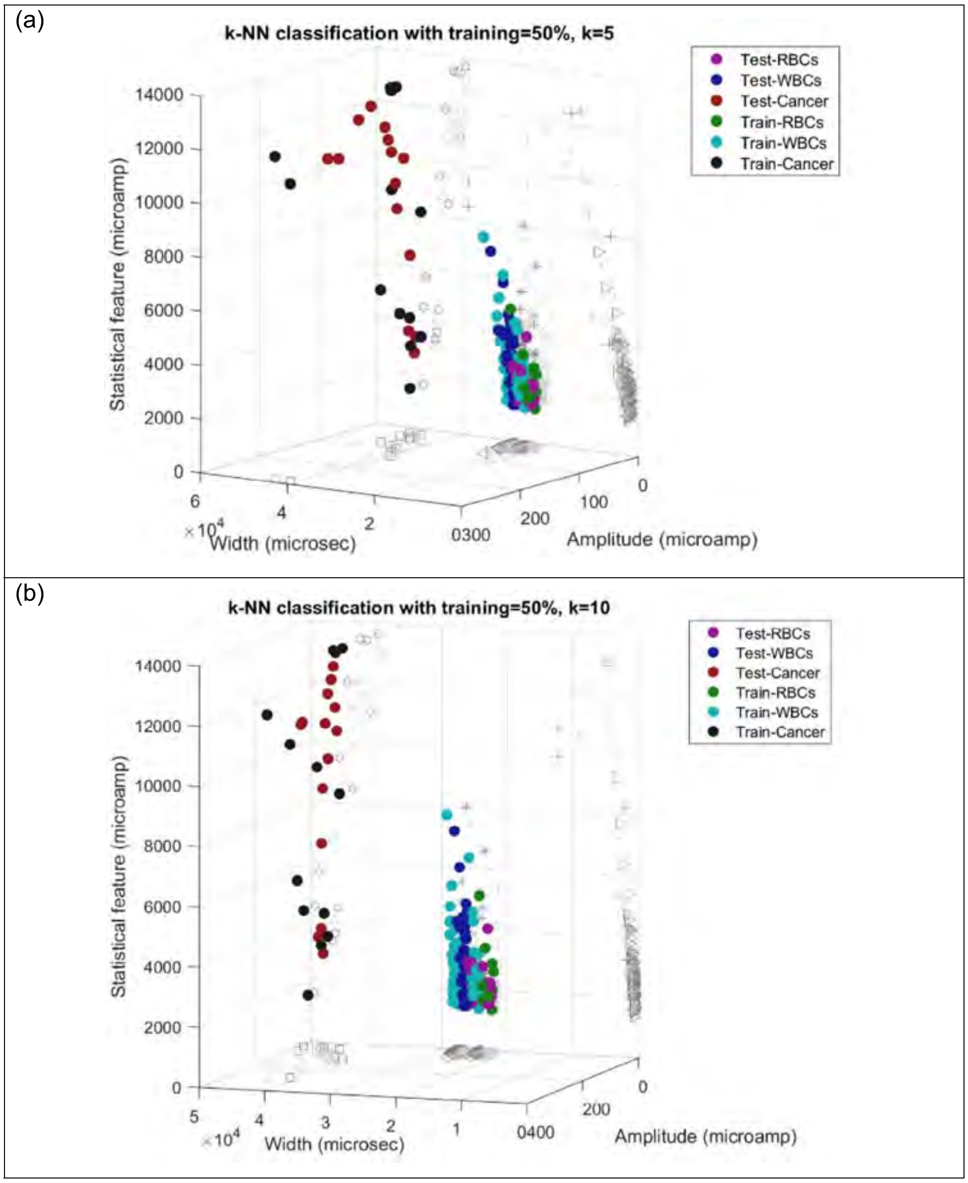

Fig. 6 – The results of *k*-nearest neighbor algorithm with training sample of 50% and (a) *k* = 5 and (b) *k* = 10.

performed on a powerful Intel Core i7-based CPU coupled with low-end Quadro-based GPU with limited memory and number of cores as compared to the state-of-the-art server-class GPUs, such as Fermi [70] and Kepler [72,79,80]. Furthermore, the detected pulses were just a few kilobytes of the acquired measurements of a typical blood sample and were significantly smaller as compared to the GPU memory, which was a value of a few hundreds of megabytes. Such a small amount of data did not significantly populate the global memory of GPU hardware, and thus, the overhead of data transfer was greater than the computation itself. Significant working set that fit in GPU memory will result in even better speedup.

Detection of diseases, in particular, cancer is an extremely time-sensitive matter. State of the art technologies and research have heavily focused on the early detection of cancer to quickly and efficiently locate the disease, and limit its effects. To aid this cause, this technique is able to increase the speed of data analysis by three to four folds for detection of cancer cells, which in turn allows an earlier diagnosis and better outcomes for the therapy. This technique is efficient, cost-effective, and does not require any state of the art technology. It is simple and easy to set-up.

The unprecedented amount of data generated from the biosensors is increasing to capture the biophysical characteristics at ever-finer granularities. Future micro/nanodevice designs are expected to include arrays to measure behaviors at extremely high resolution, and at sampling intervals of only a few nanoseconds. Therefore, this is more daunting to find useful insights into such data in real time. This inspires the need for advanced resource management mechanisms, including advanced I/O techniques, such as pre-fetching and multiple buffering coupled with advanced GPU-based setup. Kepler-based GPU setup with a significantly larger number of cores, an improved bandwidth between the cores, larger on-

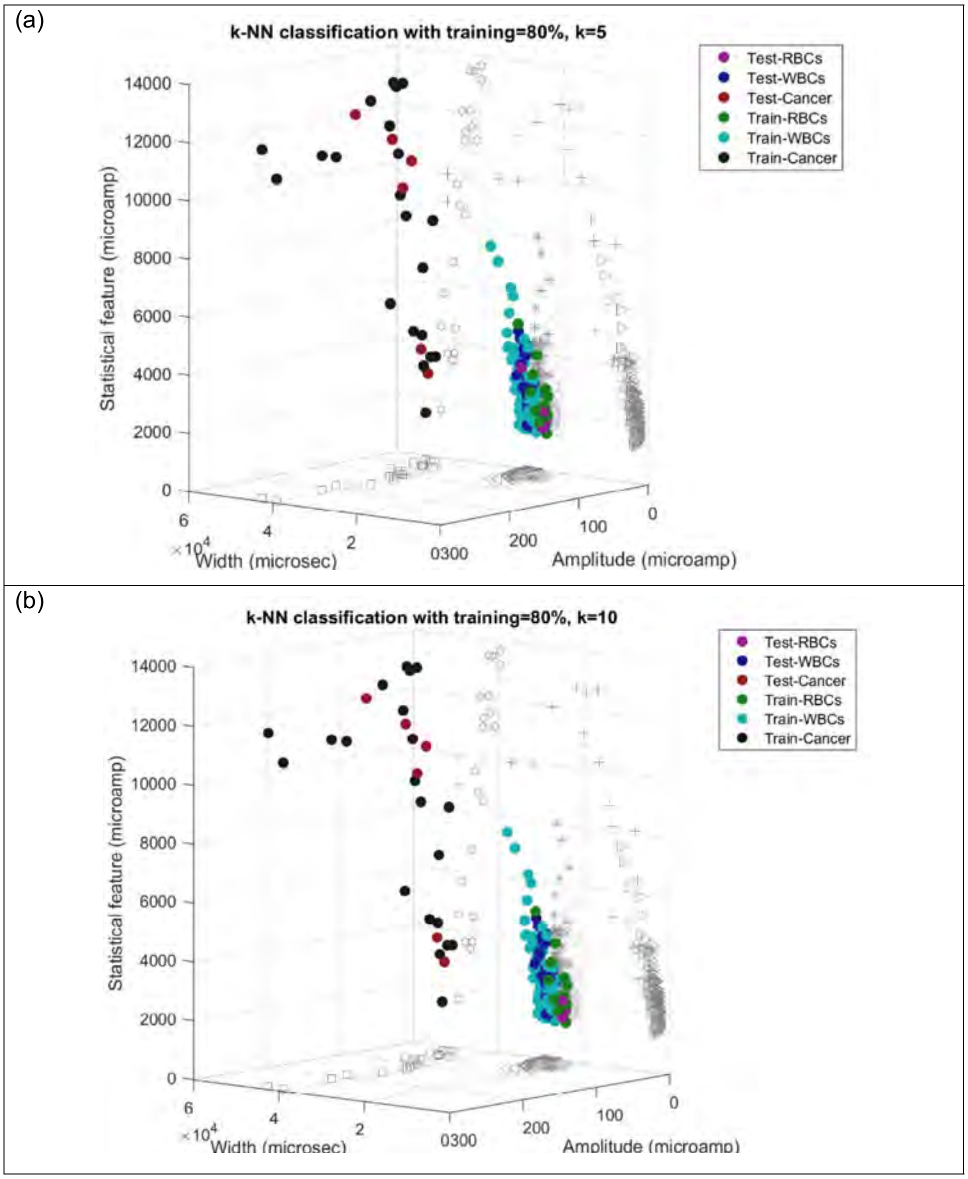

Fig. 7 – The results of *k*-nearest neighbor algorithm with training sample of 80% and (a) *k* = 5 and (b) *k* = 10.

chip memory, and more control than its ancestors seems promising in accelerating such computations. Nanoscale devices including nanopores can be the next step in this integrated machine learning technique. Additionally, the use of advanced, though compute-intensive machine-learning algorithms, such as *artificial neural networks,* and *support-vector machines,* can play a significant role to unveil the type of cancer in the target tissue.

5. Conclusions

An approach is presented to detect and classify pulses in the raw data collected from healthy and diseased cells using a machine-learning technique. The useful features were computed from the detected pulses, which were used to classify cancer clusters from the WBC and RBC clusters. The system detected cancer cells with an accuracy of 70% from a typical biological assay, which were then completely separated out from other cell clusters using their features. In addition to that, all cell types were accurately classified when training data size was increased to 80%. Using a GPU for the classification of cancer cells resulted in computations four times faster than those using a CPU.

Statistical feature computed as the average of mean and standard deviation for each pulse was used as a third metric, in addition to the pulse width and amplitude. These features completely separated out the cancer cell cluster from those of RBCs and WBCs. It, however, left the latter two clusters overlapped. This helped in labeling and detecting cancer clusters from mixed cell types with different proportions of training samples. The *k*-nearest neighbor approach detected all clusters with high accuracy when the right proportion of training samples and the right choice of the neighborhood size were

Table 2 – Accuracy of classification for 80% and 50% training sample sizes.

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

Table 3 – The performance analysis of *k*-nearest neighbor algorithms on CPU and GPU. Execution time for the main steps of *k*-nn algorithm is shown.

k-nn Algorithm	GPU-kernel execution time (msec) *k* = 5, 10		CPU time (msec) *k* = 5, 10 (Single thread execution)		
	euclid_dist	sorting_with_keys	euclid_dist	sorting_with_keys	decision_making
Training data = 25% *Test data = 75%*	0.1427	0.227	0.526	1.05	0.245
Training data = 50% *Test data = 50%*	0.217	0.275	0.525	1.06	0.18
Train data = 75% *Test data = 25%*	0.165	0.332	0.53	1.43	0.134

chosen. This work lays the foundation for automated detection and classification of biological targets that can further be used to infer useful information at even smaller scales. Nanopore data can be analyzed for disease diagnosis by using the same fundamental methods as described in this work.

Acknowledgments

This work is based upon the projects supported by the National Science Foundation under Grants CNS-1119085, CNS-1119742, CNS-1016793, CNS-1016408, CNS-1405697, and CNS-1422788. SMI was supported by National Science Foundation grant ECCS-1201878. The authors wish to thank Henry Monti and Aleksandr Khasymski for their constructive feedback on this work.

The authors would like to thank Dr. Waseem Asghar for help with data analysis. The chip fabrication was carried out at the Nanotechnology Research Center, University of Texas at Arlington, Texas. The cancer cells were shared by the lab of Dr. Robert Bachoo of the University of Texas Southwestern Medical Center at Dallas, Texas. These were obtained from consenting patients as per the approved Institutional Review Board (IRB).

REFERENCES

[1] W. Asghar, A. Ilyas, J. Billo, S. Iqbal, Shrinking of solid-state nanopores by direct thermal heating, Nanoscale Res. Lett. 6 (2011) 372.
[2] S.M. Iqbal, D. Akin, R. Bashir, Solid-state nanopore channels with DNA selectivity, Nat. Nanotechnol. 2 (2007) 243–248.
[3] W. Asghar, Y. Wan, A. Ilyas, R. Bachoo, Y.-T. Kim, S.M. Iqbal, Electrical fingerprinting, 3D profiling and detection of tumor cells with solid-state micropores, Lab Chip 12 (2012) 2345–2352.
[4] A. Ilyas, W. Asghar, Y.-T. Kim, S.M. Iqbal, Parallel recognition of cancer cells using an addressable array of solid-state micropores, Biosens. Bioelectron. 62 (2014) 343–349.
[5] W.H. Coulter, High speed automatic blood cell counter and cell size analyzer, Proc. Natl. Electron. Conf. 12 (1956).
[6] Wallace H. Coulter, Means for counting particles suspended in a fluid. U.S. Patent 2,656,508, issued October 20, 1953.
[7] S.M. Bezrukov, J.J. Kasianowicz, Current noise reveals protonation kinetics and number of ionizable sites in an open protein ion channel, Phys. Rev. Lett. 70 (15) (1993) 2352.
[8] A. Balijepalli, J. Ettedgui, A.T. Cornio, J.W.F. Robertson, K.P. Cheung, J.J. Kasianowicz, et al., Quantifying short-lived events in multistate ionic current measurements, ACS Nano 8 (2014) 1547–1553.

[9] S. Howorka, Z. Siwy, Nanopore analytics: sensing of single molecules, Chem. Soc. Rev. 38 (8) (2009) 2360–2384.

[10] J.J. Kasianowicz, J.W. Robertson, E.R. Chan, J.E. Reiner, V.M. Stanford, Nanoscopic porous sensors, Annu. Rev. Anal. Chem. (Palo Alto Calif.) 1 (2008) 737–766.

[11] A. Piruska, M. Gong, J.V. Sweedler, P.W. Bohn, Nanofluidics in chemical analysis, Chem. Soc. Rev. 39 (3) (2010) 1060–1072.

[12] J.J. Kasianowicz, E. Brandin, D. Branton, D.W. Deamer, Characterization of individual polynucleotide molecules using a membrane channel, Proc. Nat. Acad. Sci. 93 (24) (1996) 13770–13773.

[13] S. Kumar, C. Tao, M. Chien, B. Hellner, A. Balijepalli, J.W.F. Robertson, et al., PEG-labeled nucleotides and nanopore detection for single molecule DNA sequencing by synthesis, Sci. Rep. 2 (2012).

[14] W.J. Lan, D.A. Holden, B. Zhang, H.S. White, Nanoparticle transport in conical-shaped nanopores, Anal. Chem. 83 (10) (2011) 3840–3847.

[15] T. Ito, L. Sun, R.M. Crooks, Simultaneous determination of the size and surface charge of individual nanoparticles using a carbon nanotube-based coulter counter, Anal. Chem. 75 (10) (2003) 2399–2406.

[16] O.A. Saleh, L.L. Sohn, Quantitative sensing of nanoscale colloids using a microchip coulter counter, Rev. Sci. Instrum. 72 (12) (2001) 4449–4451.

[17] D. Pedone, M. Firnkes, U. Rant, Data analysis of translocation events in nanopore experiments, Anal. Chem. 81 (23) (2009) 9689–9694.

[18] J.W.F. Robertson, C.G. Rodrigues, V.M. Stanford, K.A. Rubinson, O.V. Krasilnikov, J.J. Kasianowicz, Single-molecule mass spectrometry in solution using a solitary nanopore, Proc. Nat. Acad. Sci. 104 (20) (2007) 8207–8211.

[19] J.E. Reiner, J.J. Kasianowicz, B.J. Nablo, J.W.F. Robertson, Theory for polymer analysis using nanopore-based single-molecule mass spectrometry, Proc. Nat. Acad. Sci. 107 (27) (2010) 12080–12085.

[20] A. Balijepalli, J.W. Robertson, J.E. Reiner, J.J. Kasianowicz, R.W. Pastor, Theory of polymer–nanopore interactions refined using molecular dynamics simulations, J. Am. Chem. Soc. 135 (18) (2013) 7064–7072.

[21] J. Li, M. Gershow, D. Stein, E. Brandin, J.A. Golovchenko, DNA molecules and configurations in a solid-state nanopore microscope, Nat. Mater. 2 (9) (2003) 611–615.

[22] R.M. Smeets, U.F. Keyser, N.H. Dekker, C. Dekker, Noise in solid-state nanopores, Proc. Nat. Acad. Sci. 105 (2) (2008) 417–421.

[23] J.J. Kasianowicz, S.M. Bezrukov, Protonation dynamics of the alpha-toxin ion channel from spectral analysis of pH-dependent current fluctuations, Biophys. J. 69 (1) (1995) 94.

[24] S.M. Bezrukov, I. Vodyanoy, R.A. Brutyan, J.J. Kasianowicz, Dynamics and free energy of polymers partitioning into a nanoscale pore, Macromolecules 29 (26) (1996) 8517–8522.

[25] M.A.I. Mahmood, W. Ali, A. Adnan, S.M. Iqbal, 3D structural integrity and interactions of single-stranded protein-binding DNA in a functionalized nanopore, J. Phys. Chem. B 118 (22) (2014) 5799–5806.

[26] E.C. Yusko, J.M. Johnson, S. Majd, P. Prangkio, R.C. Rollings, J. Li, et al., Controlling protein translocation through nanopores with bio-inspired fluid walls, Nat. Nanotechnol. 6 (4) (2011) 253–260.

[27] Y. Huang, S. Magierowski, E. Ghafar-Zadeh, C. Wang, A high-speed realtime nanopore signal detector. Computational Intelligence in Bioinformatics and Computational Biology (CIBCB), 2015 IEEE Conference on. IEEE, 2015.

[28] C. Raillon, P. Granjon, M. Graf, L.J. Steinbock, and A. Radenovic, Fast and automatic processing of multi-level events in nanopore translocation experiments, Nanoscale 4 (16) (2012) 4916–4924.

[29] QUB—*Software for single-molecule biophysics*. Qub.buffalo.edu. https://www.qub.buffalo.edu/, 2016 (accessed 30.03.16).

[30] Rhenley/Pyth-Ion. *GitHub*. https://github.com/rhenley/PythIon, 2015 (accessed 30.03.16).

[31] R.Y. Henley, B.A. Ashcroft, I. Farrell, B.S. Cooperman, S.M. Lindsay, M. Wanunu, Electrophoretic deformation of individual transfer RNA molecules reveals their identity, Nano Lett. 16 (2016) 138–144.

[32] X. Zhu, Z. Ghahramani, J. Lafferty, Semi-supervised learning using Gaussian fields and harmonic functions, Machine Learning Inernational Workshop then conference, vol. 20, 2003.

[33] G.D. Stormo, T.D. Schneider, L. Gold, A. Ehrenfeuch, Use of the perceptron algorithm to distinguish translation initiation sites in E. coli, Nucleic Acids Res. 10 (1982) 2997–3011.

[34] A.L. Tarca, V.J. Carey, X.W. Chen, R. Romero, S. Draghici, Machine learning and its applications to biology, PLoS Comput. Biol. 3 (2007).

[35] J. Weston, C. Leslie, E. le, D. Zhou, A. Elisseeff, W.S. Noble, Semi-supervised protein classification using cluster kernels, Bioinformatics 21 (2005) 3241–3247.

[36] C.S. Ong, A. Ben-Hur, S. Sonnenburg, B. Scholkopf, G. Ratsch, Support vector machines and kernels for computational biology, PLoS Comput. Biol. 4 (2008).

[37] Q.-H. Ye, Predicting hepatitis B virus-positive metastatic hepatocellular carcinomas using gene-expression profiling and supervised machine learning, Nat. Med. 9 (2003) 416–423.

[38] M.A. Shipp, K.N. Ross, P. Tamayo, A.P. Weng, J.L. Kutok, R.C. Aguiar, et al., Diffuse large B-cell lymphoma outcome prediction by gene-expression profiling and supervised machine learning, Nat. Med. 8 (2002) 68–74.

[39] A.C. Tan, D. Gilbert, Ensemble machine learning on gene expression data for cancer classification, Appl. Bioinformatics 2 (2003) 1–9.

[40] A.A. Alizadeh, M.B. Eisen, R.E. Davis, C. Ma, I.S. Lossos, A. Rosenwald, et al., Distinct type of diffuse large B-cell lymphoma identified by gene expression profiling, Nature 403 (Nov 2000) 503–511.

[41] W. Zong, T. Heldt, G. Moody, R. Mark, An open-source algorithm to detect onset of arterial blood pressure pulses, IEEE Computers in Cardiology, 2003.

[42] B.-U. Kohler, C. Hennig, R. Orglmeister, The principles of software QRS detection, IEEE Eng. Med. Biol. Mag. 21 (2002) 42–57.

[43] P. Du, W.A. Kibbe, S.M. Lin, Improved peak detection in mass spectrum by incorporating continuous wavelet transform-based pattern matching, Bioinformatics 22 (2006) 2059–2065.

[44] G.K. Palshikar, Simple algorithms for peak detection in time-series, in: Proc. 1st Int. Conf. Advanced Data Analysis, Business Analytics and Intelligence, 2009.

[45] L. Lamel, L.R. Rabiner, A.E. Rosenberg, J.G. Wilpon, An improved endpoint detector for isolated word recognition, IEEE Trans. Acoust. 29 (1981) 777–785.

[46] C.C. Harrell, Y. Choi, L.P. Horne, L.A. Baker, Z.S. Siwy, C.R. Martin, Resistive-pulse DNA detection with a canonial nanopore sensor, Am. Chem. Soc. 22 (2006) 10837–10843.

[47] X.S. Ling, Addressable nanopores and micropores including methods for making and using same. U.S. Patent No. 7,678,562. 16 Mar. 2010.

[48] M. Lekka, P. Laidler, D. Gil, J. Lekki, Z. Stachura, A.Z. Hrynkiewicz, Elasticity of normal and cancerous human bladder cells studied by scanning force microscopy, Eur. Biophys. J. 28 (1999) 312–316.

[49] Y.E. Choi, J.-W. Kwak, J.W. Park, Nanotechnology for early cancer detection, Sensors (Basel) 10 (2010) 428–455.

[50] J.D. Owens, M. Houston, D. Luebke, S. Green, J.E. Stone, J.C. Phillips, GPU computing, Proc. IEEE 96 (2008) 879–899.

[51] 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.

[52] R. Hussong, B. Gregorius, A. Tholey, A. Hildebrandt, Highly accelerated feature detection in proteomics data sets using modern graphics processing units, Bioinformatics 25 (2009) 1937–1943.

[53] M.C.S.C. Trapnell, Optimizing data intensive GPGPU computations for DNA sequence alignment, Parallel Comput. 35 (2009) 429–440.

[54] J.E. Stone, D.J. Hardy, I.S. Ufimtsev, K. Schulten, GPU-accelerated molecular modeling coming of age, J. Mol. Graph. Model. 29 (2010) 116–125.

[55] J.A. Cole, Z. Luthey-Schulten, Whole cell modeling: from single cells to colonies, Isr. J. Chem. 54 (2014) 1219–1229.

[56] J.E. Stone, R. McGreevy, B. Isralewitz, K. Schulten, GPU-accelerated analysis and visualization of large structures solved by molecular dynamics flexible fitting, Faraday Discuss. 169 (2014) 265–283.

[57] J.C. Phillips, J.E. Stone, K.L. Vandivort, T.G. Armstrong, J.M. Wozniak, M. Wilde, et al., Petascale tcl with NAMD, VMD, and Swift/T, in: Proceedings of the 1st First Workshop for High Performance Technical Computing in Dynamic Languages, IEEE Press, 2014, pp. 6–17.

[58] M.J. Hallock, J.E. Stone, E. Roberts, C. Fry, Z. Luthey-Schulten, Simulation of reaction diffusion processes over biologically relevant size and time scales using multi-GPU workstations, Parallel Comput. 40 (2014) 86–99.

[59] W. Shen, D. Wei, W. Xu, X. Zhu, 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.

[60] C.T. Villongco, D.E. Krummen, P. Stark, J.H. Omens, A.D. McCulloch, Patient-specific modeling of ventricular activation pattern using surface ECG-derived vectorcardiogram in bundle branch block, Prog. Biophys. Mol. Biol. 155 (2014) 305–313.

[61] S.S. Stone, J.P. Haldar, S.C. Tsao, W. Hwu, B.P. Sutton, Z.P. Liang, Accelerating advanced MRI reconstructions on GPUs, J. Parallel Distrib. Comput. 68 (2008) 1307–1318.

[62] S. Lim, K. Kwon, B.S. Shin, GPU-based interactive visualization framework for ultrasound datasets, Comput. Animat. Virtual Worlds 20 (2009) 11–23.

[63] U.C.T.D. Hartley, A. Ruiz, F. Igual, R. Mayo, M. Ujaldon, Biomedical image analysis on a cooperative cluster of GPUs and multicores, Proc. ACM ICS, 2008, 2008.

[64] D. Sato, Y. Xie, J.N. Weiss, Z. Qu, A. Garfinkel, A.R. Sanderson, Acceleration of cardiac tissue simulation with graphic processing units, Med. Biol. Eng. Comput. 47 (2009) 1011–1015.

[65] H.E.R.C. Men, X. Jia, S.B. Jiang, Ultrafast treatment plan optimization for volumetric modulated arc therapy (VMAT), Med. Phys. 37 (2010) 5787–5791.

[66] T.Y. Huang, Y.W. Tang, S.Y. Ju, Accelerating image registration of MRI by GPU-based parallel computation, Magn. Reson. Imaging 9 (2011) 712–716.

[67] Y. Zhuo, X.L. Wu, J.P. Haldar, T. Marin, W. Hwu, Using GPUs to accelerate advanced MRI reconstruction with field inhomogeneity compensation, in: GPU Computing Gems, Emerald Edition, Elsevier, 2011.

[68] A. Hafeez, W. Asghar, M.M. Rafique, S.M. Iqbal, A.R. Butt, GPU-based Real-time Detection and Analysis of Biological Targets using Solid-state Nanopores, Medical & Biological Engineering & Computing 50 (6) (2012) 605–615.

[69] Nvidia.com. (2016). Parallel Programming and Computing Platform | CUDA | NVIDIA | NVIDIA. [online] Available at: http://www.nvidia.com/object/cuda_home_new.html (accessed 20.06.16).

[70] J. Kurzak, S. Tomov, J. Dongarra, Autotuning GEMM kernels for the Fermi GPU, IEEE Trans. Parallel Distrib. Syst. 23 (2012) 2045–2057.

[71] G. Zumbusch, Tuning a finite difference computation for parallel vector processors, 11th Int. Symp. Parallel and Distrib. Comput. CPS, IEEE, pp. 63–70, 2012.

[72] J. Lai, A. Seznec, Performance upper bound analysis and optimization of SGEMM on Fermi and Kepler GPUs, in Code Generation and Optimization (CGO), 2013 IEEE/ACM International Symposium on, 2013, pp. 1–10.

[73] R.M.K.J.E. Hopcroft, A n5/2 algorithm for maximum matchings in bipartite graphs, SIAM J. Comput. 2 (1973) 225–231.

[74] J.C. Sancho, D.J. Kerbyson, Analysis of double buffering on two different multicore architectures: Quad-core Opteron and the Cell-BE, International Parallel and Distributed Processing Symposium (IPDPS), IEEE, Los Alamitos, 2008.

[75] S. Few, Data visualization for human perception, Encyclopedia of Human-Computer Interaction, 2010.

[76] 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.

[77] B. Nichols, D. Buttlar, J. Farrell, Pthreads programming: A POSIX standard for better multiprocessing, O'Reilly Media, Inc., 1996.

[78] W. Asghar, Y. Wan, A. Ilyas, R. Bachoo, Y. Kim, S.M. Iqbal, Electrical fingerprinting, 3D profiling and detection of tumor cells with solid-state micropores, Lab Chip 12 (2012) 2345–2352.

[79] N.-P. Tran, M. Lee, S. Hong, D.H. Choi, Multi-stream parallel string matching on Kepler architecture, in: Mobile, Ubiquitous, and Intelligent Computing, Springer, 2014, pp. 307–313.

[80] R. Salomon-Ferrer, A.W. Götz, D. Poole, S. Le Grand, R.C. Walker, Routine microsecond molecular dynamics simulations with amber on GPUs. 2. Explicit solvent particle mesh Ewald, J. Chem. Theory Comput. 9 (2013) 3878–3888.