Research Article

Impact of Graphics Processing Units on Diagnosis using Bio-Imaging

Abdul Hafeez[1]*, Akhtar Nawaz Khan[2] and Zahid Ullah[3]

[1]Department of Computer Science and IT, Jalozai Campus, UET Peshawar, Khyber Pakhtunkhwa, Pakistan; [2]Department of Electrical Engineering, Jalozai Campus, UET Peshawar, Khyber Pakhtunkhwa, Pakistan; [3]Pak-Austria Fachhochschule: Institute of Applied Sciences and Technology, Haripur, Pakistan.

Abstract: Important biological molecules including DNA/proteins and diseased human cells can be detected by a variety of micro and nanoscale devices and systems. Unfortunately, these biomedical applications suffer from huge amount of data. That is, the data generated by such systems is so large that a typical computer workstation cannot handle it in real-time. Traditional approaches rely on unloading raw data to off line storage and suffer from the trade-off of an insufficiently rigorous sampling rate. Furthermore, the phases leading to useful decision-making often follows pre- and post-processing in a subjective manner, which is tedious, time-consuming and erroneous. An important and an unrealized need of these systems is the real-time answer from the acquired data, which is often multi-dimensional and at remarkably high-resolution. The inherit noise in the data is another major bottleneck for real-time processing. This entails expediting the decision-making process as well as to instill intelligence in the approach to reduce subjectivity to a greater extent. To harness the high computational power offered by commodity graphics processing units (GPU) and accomplish the decision-making in real-time has gained traction in a gamut of recent biomedical applications. This review article investigates the use of state-of-the-art GPUs to benefit the biomedical community through real-time decision-making from huge amount of biomedical data mainly due to its massively parallel architecture. The recent advances in GPUs such as Fermi- and Kepler-based setup seem to be promising for an increasing complexity in the algorithms and the demands of growing data in biomedical applications both in dimension as well as in throughput. Understanding these approaches can help the biomedical community to shift batch-mode processing from cluster-based machines to *real-time processing* on low-end machines or smartphones in order to help the decision-makers and stack holders for early disease detection.

Received: July 13, 2020; **Accepted:** October 13, 2020; **Published:** December 19, 2020
***Correspondence:** Abdul Hafeez, Department of Computer Science and IT, Jalozai Campus, UET Peshawar, Khyber Pakhtunkhwa, Pakistan; **Email:** abdul.hafeez@uetpeshawar.edu.pk
Citation: Hafeez, A., A.N. Khan and Z. Ullah. 2020. Impact of graphics processing units on diagnosis using bio-imaging. *Journal of Engineering and Applied Sciences,* 39(2): 143-153.
DOI: http://dx.doi.org/10.17582/journal.jeas/39.2.143.153
Keywords: GPU, Real-time data mining, Medical imaging, Neuronal recording, Solid-state pores

Introduction

In addition to the traditional approaches, the design of novel systems and techniques at meso, micro, and even nanoscale has sparked new methods in order to explore and analyze the molecular origins of critical diseases. Miniaturize electronics devices atop these endeavors has triggered the measuring of molecular interactions using resilient approaches that can precisely transduce biotic interactions at a rudimentary scale. Such profound attempts promise to leverage instruments appropriate for diagnosing diseases at an early stage, scan prognosis in a finer detail, and follow and predict the drift of disease evolution in counter to the state of the art medicine and novel remedies. However, these approaches suffer by one-way or the other from large data sets. Conventional approaches including mass spectrometry (MS), multi-electrode arrays (MEA), and magnetic-resonance imaging (MRI) generate multi-dimensional and high-throughput data. The quantity of the measured data per unit time has progressed on the order of magnitude i.e., exponentially in resolution as well as in dimensionality, partly because of high-spatial resolution as well as finer discretization of the measured signals/images. Therefore, it has set off almost impossible to correctly distill interesting patterns in the arising real-time streaming data using traditional approaches.

The emerging accelerators and heterogeneous-based computing especially graphics processing units (GPUs) have shown huge potential in accelerating biomedical applications. Epidermal Electronics System (EES) is one such direction, a breakthrough research for measuring EEG, ECG, and EMG. The performance for measuring these activities is up to the par so far. With the passage of time such system will expand them to measure other essential human activities. This will lead to an increase in the fabrication per unit through advanced micro and nanofabrication approaches resulting in generating large amount of data. Such data deluge will incur delays in the detection of diseases and will not produce results at the right time and hence the processing will never happen in real-time. The delayed diagnosis of the patients can further lead to the death in diseases that require an immediate attention such as cancer.

In fact, handling a significant amount of data is non-trivial and entails significant computing resources. More specifically, the huge data streams created by MS, MEA, Nanopore measurements, usually using a sensor network entails innovative solutions to efficiently analyze and compute the target results. An advantageous remedy in the discussed design space may hinder the implication of traditional resources as shown in Figure 2. This makes the processing and decision-making never happen in real-time. To this point, it is crucial to intelligently automate the process of data analytics owing to an accurate and effective prognosis of a target disease.

The recent shifts has shown a tremendous growth in the use of number-crunching accelerators i.e., GPUs for computational-hungry and data-intensive problems, especially to the said biomedical applications (GraphStream, 2006). GPU overwhelms the computations of statistics in crucial bio inspired applications encompassing big data. Furthermore, these devices coupled with Microsoft Direct 3D leverages the computation of huge finite element methods (Zhang *et al*., 2018). It was originally designed for gaming purpose; however, later real scientific domains espoused it owing to the inherit massively parallel architecture. GPU can perform graphics operations at the orders of magnitude faster as compared to a general purpose CPU due to its architectural design. This is happening because of hundreds and even thousands of cores hosted on a single GPU chip that can perform such operations incredibly fast. The number of threads is also increased up to thousands due to the incremental improvements in the GPU core technology – fueled by the tremendous demands of performance-critical applications.

Recent trends in novel standalone GPU and GPU cluster designs (NVIDIA 2009, 2012) setup support dynamic parallelism, multiple kernel launches, and all standard integer and floating-point computations supported by traditional CPUs, and a shift towards general-purpose computation. Such drastic improvements are a step towards enabling GPU an ideal chip for scientific and engineering applications with a significant amount of parallelism. Tremendous parallelism is required in order to map and address the market needs of increasing data- and compute-intensive workloads. Benchmark workloads have been provided by the GPU community such as MGPU Sim, MLPerf and state of the art benchmarks those address the concurrency requirements in order to address the simulation requirements for a range of domain applications with a configurable GPU architecture based on user requirements against a complementary CPU machine (John *et al*., 2008). Examples of such computationally and data intensive applications include acceleration of molecular dynamics such as continuous and constant pH molecular dynamics (Bryer *et al*., 2019), profiling simulation dynamics of largescale molecules (Allec *et al*., 2019), and a heterogeneous based setup for the simulation of important and large chemical and biological molecules for an improved performance (John *et al*., 2010).

Furthermore, GPUs have been incorporated for an improved simulation and uncovering the underlying physics of interactions happening between residues pertaining to differing families of proteins (Haldane *et al*., 2021), accelerating the prediction of important proteins absolute/relative interactions and binding phenomenon for crucial drug design (He *et al*., 2020), accelerated computation of features in proteomics data sets that is generated by MS (Hussong *et al*., 2009), detecting cardiac arrhythmias and critical cardiac disorders in electric current in real time that stems from the relaxation and contraction behavior of heart muscles and hence, significantly reducing the death rates by detecting and addressing the disease at an early stage (Sehgal *et al*., 2019), simulating cardiac at a tissue level that involves calculations of billions of partial differential equations in merely few milliseconds (Vasconcellos *et al*., 2020) simulation of cardiac tissue in real-time (Sato *et al*., 2009), accelerated simulation of ECG (Shen *et al*., 2009), GPU-accelerated MRI reconstruction algorithms, for instance, reconstructing a human brain in order to blend the orientations up to a fiber level using Bayesian based estimations, using monte carlo based simulations to model MRI as well as scatter corrections involved in a PET for providing quantitative and quantitative information (Zhuo *et al*., 2011), separating overlapping spikes recorded from neuronal activity (Schofield *et al*., 2019), a software based model i.e., Neuro GPU for simulating neurons and its activities on GPU hardware (Ben-Shalom *et al*., 2020) and collateral mining of the spike streams stemming from neurons in an MEA experiment (Cao *et al*., 2010).

Specifically, in biomedical computing domain GPUs can become computational biochips instead of standalone computational chips due to its pervasive adoption in the domain. Furthermore, efforts have also shown significant improvement in the performance by coupling GPU processing with enhanced I/O techniques in order to overlap the communication with the computation for an accelerated computing (Rafique *et al*., 2009, 2010).

This review discusses the hardware and software architecture of GPUs followed by the examples of biomedical applications suffering from high-dimensional and noisy data and that entails processing in real-time. We further survey efficient use of GPUs for automating the process of finding interesting patterns in real-time in these applications.

Overview of graphic processing unit (GPU)

GPUs are specialized electronic chips originally designed for graphics and gaming applications. However, due to their highly parallel architecture, they are now adopted by scientific community for massive scientific computations. GPUs are typically coupled with CPU as a co-processor to accelerate the data-parallel and compute-intensive jobs. As compared to CPUs, GPUs are very efficient in processing embarrassingly parallel applications e.g., computer graphics, image processing, ray-tracing, computational fluid dynamics, molecular dynamics, cardiac simulations, bioinformatics etc. The performance-critical and data-parallel portions called hot-spots are identified in applications and accelerated by offloading to the GPUs. Normally, the control code consisting of branches and conditions is executed on CPU. This hybrid computing with control-code on CPU and data-parallel code on GPU results in significant performance improvement. The CPU typically has 2, or 4 cores, while GPU has hundreds of cores on chip as shown in Figure 1.

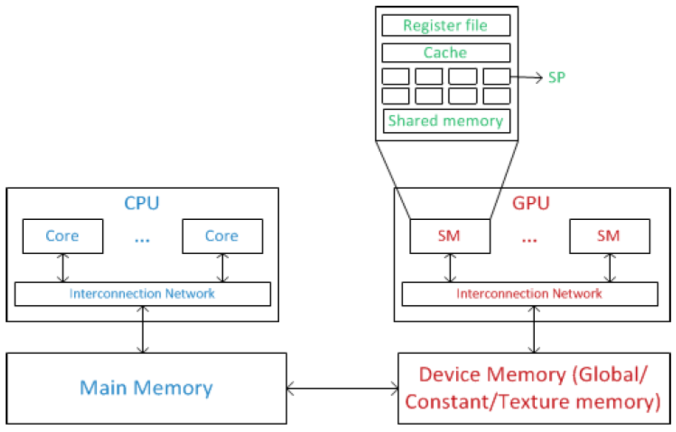


Figure 1: *CPU coupled with GPU: The data is transferred from the primary to the device memory. After kernel-run on the device, the results transcript to main-memory for visualization purpose.*

CUDA programming model

The CUDA kernel comprises of *lightweight* threads classified in blocks, which are further batched into a grid kernel as illustrated in Figure 2. Typically, a block can support up to a maximum of X (512 etc.) threads and a minimum of 64 threads. In newer GPUs such as 1080Ti that has 3584 cores can run up to a maximum of 16x3584 threads concurrently, where as one GPU core can run 16 threads. A batch of 32 threads constitutes a thread block that is called a warp. Furthermore, shared memory which is an expensive memory is basically available to the entire threads within a block and thus, synchronizing the entire threads in that particular block. In contrary, the threads residing in different blocks do not encourage sharing of the *shared memory* and therefore, prohibits synchronization across the blocks. The entire thread set within a kernel launch has the right to access global memory. In a branch divergence issue, the threads within a block split apart and crunch a chunk of code, while another portion of the threads consume another piece of the code. In such cases, synchronization per thread block i.e. sync threads is ultimately required. Such synchronization ensures that the whole threads in a particular block are consummated and have attained an ordained barrier.

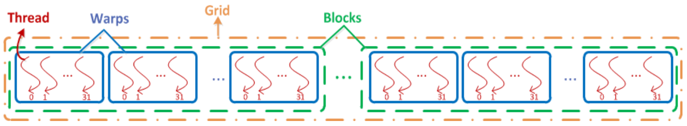


Figure 2: *CUDA thread hierarchy. Threads are grouped into warps. Each block contains at least two warps. Blocks are further grouped into Grids.*

Furthermore, the programmer has the freedom to tweak the execution configuration pertaining to a kernel by altering dimensions of a block. The indexing scheme designed for threads pertaining to higher dimensions becomes $x+yD_x$ and $x+yD_x+zD_y$ i.e., for 2-dimensions (D_x, D_y) and 3-dimensions (D_x, D_y, D_z) size of blocks, respectively. Regardless of the dimensionality issue, the entire threads do not dare to cross the maximum threshold set off by the hardware architecture. This features enables launching of different execution configurations in order to map a gamut of target applications on a GPU for an enhanced performance. Legacy architectures with capability < 2.0 such as Tesla-based GPUs only allow one kernel launch at a given time. Collaborative kernels organize themselves successively and run independently on a GPU. However, multiple kernels can be launched to execute collaterally as shown in Figure 3. Furthermore, new features allow altering an execution configuration on fly without the interference of a CPU and therefore, facilitate *dynamic parallelism*.

For instance, consider the example of a simple squaring function. To efficiently execute this on GPU, we assign each thread a single item to square it. All the threads compute their respective multiplications required for squaring concurrently in SIMD fashion. Once all the multiplications are finished, the threads are synchronized before sending results back to the CPU for display purposes. The notion of assigning each multiplication to a single thread is similar to loop unrolling, because it's a loop that computes all the multiplications involved in the squaring function on the CPU, and here we break the overall computation into concurrent operations by distributing it among multiple threads to execute in parallel instead. Hence, all the threads crunch same piece of code at the same time.

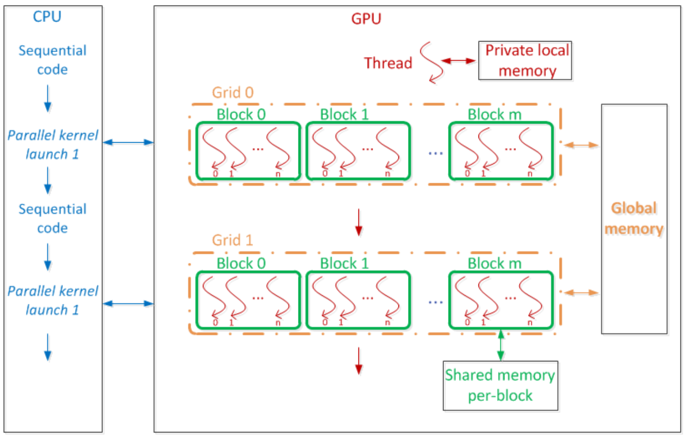
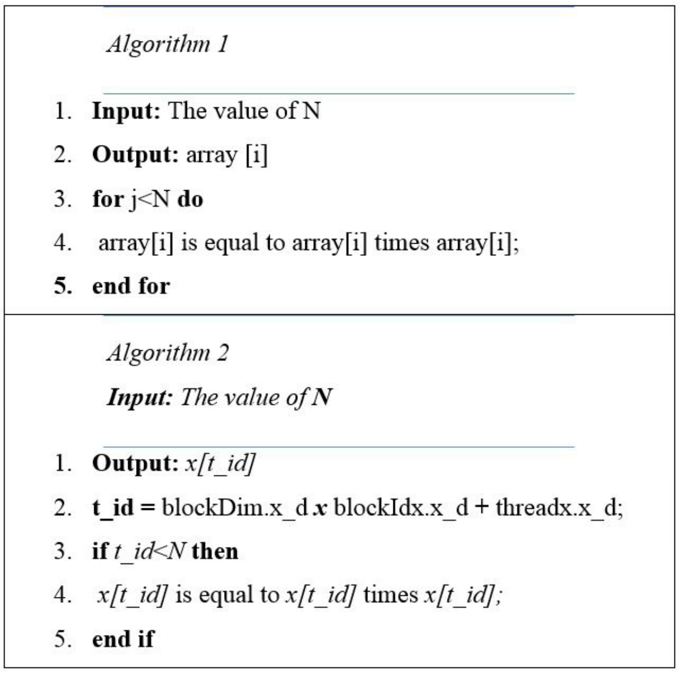


Figure 3: *Multiple kernel launches on the GPU. Different combination of the # of thread blocks and # of blocks per grid can be used for multiple kernels.*

Table 1 illustrates this example by showing the sequential loop on the CPU, and its CUDA version on the right hand side. The square function array takes an array of length N; and multiplies them individually to find the squares of all the data items in the array.

Table 1: *CPU sequential code vs. GPU code for implementing the dot product.*



```
Algorithm 1
1. Input: The value of N
2. Output: array [i]
3. for j<N do
4.  array[i] is equal to array[i] times array[i];
5. end for
```

```
Algorithm 2
Input: The value of N
1. Output: x[t_id]
2. t_id = blockDim.x_d x blockIdx.x_d + threadx.x_d;
3. if t_id<N then
4.  x[t_id] is equal to x[t_id] times x[t_id];
5. end if
```

However, its parallel counterpart in CUDA spawn threads equal to the number of multiplications i.e. N, such that each thread corresponds to a particular item of the input array. Note that, millions of threads can be launched on current GPU architecture. However, programmers can constraint the number of threads to N for smaller value of N (few hundreds). Also, this number is limited by the usage of underlying hardware resources discussed in the GPU architecture section. To get the corresponding position of the elements in the array; the number of threads is multiplied with the block index, and added to the local index of the thread within the block. Let's suppose the size of each array is N = 4096 items, and the # (blocks) = 16, and the # threads/block, block Dim.x is 256. The block Idx.x keeps track of the individual block ids, while threads Idx.x is the index of the individual thread within the block. To access 1000th item in the input arrays, the thread index becomes 3*256 + 232 according to the equation of tid shown in Table 1; where block Idx.x is 3; block Dim.x is 256 and thread Idx.x is 232; equals to 1000.

GPU architecture

The scalable streaming multiprocessors (SMs) hosting at least eight streaming processors (SPs) also called cores constitutes the underlying hardware of a GPU, as highlighted in Figure 3. The warp which is batch of 32 threads that can run concurrently is scheduled to execute on SMs, in SIMD fashion. However, scheduling of warps on SMs is not exposed to the programmer. CUDA schedules multiple blocks to run concurrently on same SM. The space of shared memory and aggregate of registers determines the maximum number of concurrent blocks that can be scheduled on an SM instantaneously. The scheduler makes sure that the whole set of threads in a block gain access to the required resources before scheduling it on an SM. The blocks executing on an SM at any instant are referred as *active blocks*.

The shared memory is implemented as an SRAM with an SM, and has very low latency, almost as registers. However, global memory comes as an off-chip DRAM and has high latency as compared to shared memory. Additionally, GPU has read-only memories: constant and texture memory as an off-chip accessible by all the threads in a kernel launch. Constant memory is used for storing kernel parameters in newer architectures such as Fermi. Texture memory offers different addressing modes and can be used for 2-D pre-fetching for 2D texture memory etc.

Impact of GPUS in crucial biomedical applications

GPUs have been an effective solution for parallelizing the compute-intensive jobs in biomedical applications. The example applications are discussed below.

Demand of acceleratoin in mass spectrometry analysis: An analytical technique that measures the expression analysis of proteins from charge to mass ratio of ions constitutes mass spectroscopy (MS). The technique basically provides information about the molecular formula and weight and the relative proportion of hydrogen, oxygen and carbon in a target compound. Such approach has several applications in medical diagnostics to differentiate between the sick and healthy, biomedical engineering, and therapy. MS further helps in unveiling the mass, structure, and composition of the molecules under study ranging from pure samples to much complicated mixtures e.g., peptides which are digested proteins. Furthermore, the technique helps in decision-making such as finding the presence of a disease. The ionization and acceleration of peptides within an MS helps in determining the inherit fingerprint by analyzing the intensity against mass.

The mass to the charge ratio, y-axis that portray the intensity along with the z-axis as additional parameters produce the output as 3D contours for further analysis. The problem with MS is that it generates larger streams of data and can quickly overflow hundreds of gigabytes. GPU-based implementation provides a faster approach for automating the feature detection using adaptive wavelet transform (Hussong *et al*., 2009). The ideal case for the results is higher precision without sacrificing the performance.

Convolution is naturally a parallel operation and therefore, enables the algorithm to map well enough to the underlying parallel architecture. The number of threads that is proportional to the data points performs convolution concurrently in CUDA. The algorithm intelligently splits the map (an input of scan points) such that each piece contains 512 points. Part of the reason is that the conventional architectures support up to a maximum number of 512 threads/block. An individual thread ascertains the convolution operation on a corresponding itemwith respect to the neighborhood items. The goal is to load additional points into the shared memory for facilitating convolution of the intensities that are in the neighborhood of the assigned point per thread. It is pertinent to mention that the points in the neighborhood fetched for a specific convolution of data point cannot be computed in prior. This difficulty in computing mainly arises due to the irregular spacing in an MS data. In safe mode, a maximum quantity of the memory is dedicated to load the whole associated points in the neighborhood that is based on the maximum mass.

The author has implemented two GPU-based implementations with varying compute and memory capabilities. The counter implementation on CPU uses 8 cores tuned at 2.3 GHz with 16GB main memory. The simple implementation on GPU loads spectrum in texture memory and shows significant speedup. An implementation uses an on chip shared memory for storing and computing spectrum data further accelerates the computation. Such approach even with the lack of approximations is faster without sacrificing better quality results, resulting in 200X speedup using real-world protein data in contrast to the analogous code on CPU. Furthermore, implementation on a contemporary device produces an acceleration up to 10X faster. These analyses demonstrate that analyzing larger scale proteomics entails incorporating advanced GPUs.

Recent research such as GPU-DAEMON illustrates a speed of 386 times over CPU-version and an acceleration of up to 50 times over previous GPU version. GPU-DAEMON aims at achieving maximum occupancy on GPU hardware, an optimized algorithm, an efficient use of memory. Combined these approaches result in an improved performance. Furthermore, three cases studies stemming from GPU-DAMEON in the form of GPU-ArraySort, G-MSR and GPU-PCC has been presented.

Accelerated medical imaging

Magnetic Resonance Imaging (MRI) provides quantitative and definitive insights for medical imaging, especially in detecting tumors in human brain and monitoring the functioning of heartbeats. MRI has the ability to study cancer and produces better images in order to contrast between the different soft tissues of the body as compared to the CT–Computed Tomography or radiographs. The data acquired from clinical trials need to reconstructed for analyzing disease progression with minimal losses in an image quality. The automation of MRI imaging is not so straightforward mainly because of the imaging artifacts, higher frequency noise, and an extended acquisition time. Furthermore, the scanning trajectory that can be either Cartesian or non-Cartesian has a profound effect on the reconstructed radiographs quality. The scanned trajectory can be either cartesian or non-cartesian. Nonetheless, a non-Cartesian trajectory is preferable in MRI because of the resiliency it offers to noise. Such reconstruction computation is trivial; however, it does not inculcate the anatomical information (Halder *et al*., 2008). The anatomical information used for an optimal image reconstruction (Fessler *et al*., 2005; Pruessmann *et al*., 2001; Sutton and Fessler, 2003) helps in reducing the noise and achieve a higher SNR per scan while does not affect the resolution of the target image features. Sophisticated reconstruction algorithms devised for scalable problems in clinical trials and disease progression can be accelerated on GPUs for a real time solution. A typical non-uniform scan in a reconstruction technique consumes merely 2 minor even less on a GPU-based implementation as compared to its counter CPU code that takes approximately 23 minutes as captured by Stone *et al*. (Stone *et al*., 2008). These advanced reconstruction algorithms are faster as well as more precise i.e. the GPU produces 12% error as compared to the analogous CPU code that results in an error of 42 %. The implementation of reconstruction algorithm on GPU varies from a naive implementation with zero optimizations to a little optimized version and even well-tuned implementations that incorporates the best match of loop unrolling by a factor of 5, tiling 2048 scan points in an individual thread and 320 the number of threads/block for the target application.

Note that these parameters vary from application to application depending upon data-parallel nature of the application. In the given case, such tuning tremendously reduces the aggregate of accesses to the global memory, which further mitigates the off-chip memory bandwidth and therefore, results in an improved performance. More specifically, the runtime reduces to 59 seconds. In comparison, running well-tuned version on a multi-GPU setup results in an accumulated runtime of 18 sec. It is pertinent to mention that a floating-point computation of single precision along with trigonometric operations that are approximated provides an improved performance and less error in the real image. However; there is a better chance to degrade the output. Enhancing the standard of the acquired images along with an accelerated implementation will increase the throughput of the scanner and thus will increase the patient comfort. Interestingly, a speed up from 2X to 9X has been attained for the said reconstruction algorithms (von Rymon-Lipinski and Keeve, 2004; Sumanaweera and Liu, 2005; Schiwietz *et al*., 2006). IBM technology group have used Cell BE to accelerate MRI reconstruction up to 17 times.

Recent efforts show an acceleration ranging from 28.81 – 89.12 with an increasing size of images from 128 x 128 – 1024 x 1024 without sacrificing the performance pertinent to the segmentation. Achieving an optimal performance for automated image registration algorithms is crucial to MRIs. The automated image registration algorithm computes an optimal mapping for the source image against the reference image through similarity metrics and transformations. These metrics are further optimized in order to compute an optimal transformation based on the similarity metrics. Unfortunately, these registrations involve outstanding iterations and therefore, are deliberate even on a typical workstation. Such behavior obstructs the post-processing of medical images for useful decision-making. GPU-based MRIs approaching to 14-fold acceleration for image registration techniques has been reported (Huang *et al*., 2011).

GPUS in electrocardiography (ECG)

ECG is a gold standard for monitoring and recording the electrical activity pertaining to a human heart. It provides little discomfort to the patient and is more ubiquitous than MRI and CT scans. Furthermore, the technique captures the activity of heart in 3D / 4D views. The raw data acquired from these applications is higher in volume and pose challenge when rendering especially the images with higher dimension as those are computationally expensive.

GPUs underlying parallel architecture can be exploited to render images of higher dimensions from the sampled ultrasound data and attempts to approaches to the speed that is required by a human heart (GPU Gems). GPUs have shown 13.03 times faster computations to its sequential counter code (Shen *et al*., 2009). However, the demand for an increasing amount of data needed in GPU computation hampers the additional acceleration. The sophisticated GPU setup can cause a revamped performance due to an increased number of cores, larger spaces of memory, and an increased bandwidth of memory that bridges between the device and CPU.

In a smartphone-based approach for a quicker and easier detection of ECG, a speed of about 6x is achieved on average, while a maximum speed up of 23x is reported, especially in artifact removal. The study utilizes a qualcomm based processor i.e., Snapdragon 820 coupled with a GPU of Adreno 530 with a memory of 6 GB. Such processor hosts two cores at 2.15 GHz, while the GPU supports 256 arithmetic and logic units that are clocked at 624 MHz optimized for parallel computation.

Ray casting is another technique that is frequently used for rendering the volume. This technique aims to project 2D data into a 3D plan through tracing out the rays from a viewpoint into a viewing volume. A volume rendering framework for ultrasound datasets that incorporates GPU-based volume rendering has been developed (Lim *et al*., 2009).

Automated analysis of multi-electrode neuron action potentials

EEG, fMRI, and MEAs are techniques aimed for recording and analysis of the neuronal activity of human brains, but the problem with these approaches is that they do not allow to record single neuron activity. To record a neuronal activity up to the granularity of a single neuron, a technique called single-unit recording is used. Single-unit recording measures electrophysiological activities of a single neuron through microelectrode system. This allows measuring of the intra- and extra-cellular phenomenon of a human brain, the action potentials. The measured information can be applied to Brain Machine Interface (BMI) to record the brain activity (action potential) and decode it into the intended response, which can control the movement of an external device e.g., prosthetic limb in assistive technologies, or computer cursor. Cao *et al*. (2010) has focused on the use of data mining algorithms towards the neuronal event streams stemming from the MEA to analyze the human brain.

The interesting firing patterns of the spikes in neuron action potential gives cognizance into the underlying cellular level activity of a neuronal tissue. The challenge in analyzing such useful insights are the sizable event streams accumulated from an MEA and a classical 64 channel MEA can simply end up in a couple of million pulses of data merely in a few seconds. Furthermore, these experiments can keep running for months resulting in hundreds of millions of neuronal action potentials. Analyzing these larger datasets entails high-end data storage and computing resources. Not to mention, extracting useful patterns in the mentioned data in real-time requires dimensionality reduction and intelligent classification techniques.

The goal is to concurrently compute the frequency episodes between non-overlapped occurrences in the event streams. One thread per Occurrence fulfills the requirement of mining lesser episodes, while the two-pass elimination counts the larger quantity of episodes. The problem with the former approach is that it utilizes GPU much below its potential due to the dependencies in the data and the accompanied branch divergence that serializes the code and hence making it harder to be parallelized. The later approach assigns one episode per thread. If there are many episodes, GPU can be utilized to its full. Based on the query of length 5, which needs 97 byte of registers and 220 bytes of shared memory, merely 32 threads can be schedule on a GPU, which hosts only 16KB of registers and the shared memory. The elimination of unsupported episodes followed by the complex state machine (two-pass elimination) to mine the supported episodes, not only improve the performance but also results in less time complexity as compared to the overall complex state-machine based algorithm (hybrid algorithm). Since local memory can be used as alternate to registers and is stored in global memory, which is accessed frequently and hence, incurs larger latency. This also comes from the fact that hybrid approach uses 80 bytes of local memory and 17 registers, in contrast to the elimination algorithm, which needs 13 registers without any local memory. Moreover, for greater number of episodes; the number of memory accesses are least by the two-pass elimination technique over the hybrid approach. Finally, the algorithms are evaluated on an NVIDIA GTX280 GPU.

Another study that stresses a real-time analysis include the processing of large neuronal data emerging from assistive technologies (Romero-Ortega *et al*., 2009). The duty cycle needed to capture the activity of neuronal action potential ranges from a section to a couple of milliseconds (Lewicki, 1998). This requires a sampling rate of fewer kHz. Considering a resolution of 1 byte that can lead to 256 distinct voltage levels for an individual electrode, the system must have the ability of measuring at least at 1 million Bytes/sec. The necessity for adapting to higher rates of potentials can easily overflow fewer GB per second due to the agglomerated rates. The conventional techniques that encompass fewer electrodes trust on pushing raw data in storage devices for batch processing, which costs longer turnabout pauses from the actual collection of data to the excitation of neurons for generating important electrical stimulants. Such longer delays are unacceptable for assistive technologies. To this end, we need high-performance GPUs for real time neuronal action potentials.

Along those lines, a recent study uses a GPU-revamped simulator based on neural networks approach to simulate a human brain with a developed tool Brain2GeNN. This study demonstrates a maximum speed up of more than 200 fold compared to a standalone CPU core, while simulating one million neurons based on Hodgkin-Huxley (Stimberg, 2018). For other models, GeNN shows differing speedups.

Conclusions and Recommendations

The speedup achieved in different applications with the use of GPUs: 200x over real data-sets in mass spectrometry; making MRI reconstruction algorithms up to 9X faster; and a speedup of more than twice for ECG calculations; suggests that inherent parallelism in such biomedical applications matches the massive parallelism of GPU devices and provides naturally high-performance computing platforms to compute results for such applications in real-time for decision making. The speedup achieved with GPU in pore application is fascinating. Massive loop-level parallelism has the potential to harness full computational power of GPUs. The more inherent parallelism an application has, the more acceleration it gets with GPUs. Unfortunately, the branch divergence and data dependency in code result in de-acceleration on GPUs; as such behavior makes the code sequential instead. This ideally suggests running conditional and control code on CPUs and off-loading the data-parallel and compute-intensive jobs to GPUs.

The GPUs are promising for the biomedical applications but offers several challenges. The programming effort required to learn CUDA-style coding itself poses a challenge. Furthermore, the application development involves learning usage of different types of memories on GPU, and the I/O communication involved between the CPU and GPU. One key challenge is the limited memory and I/O resources of GPU. The cost of off-loading larger data-inputs on GPU is non-trivial, and in some applications frequent memory transfers incurs significant pressure on GPU memory, and can choke the memory bandwidth between CPU and GPU. One way to mitigate such memory pressure is to divide the data into smaller chunks, and then stage the data to the GPU memory for processing. This data chunk shouldn't be too small to increase the frequency of data transfers, leading to less occupancy on GPU cores; and also, shouldn't be too large to increase the time per transfer and making the GPU processing cores idle most of the time. This need benchmarking on different size of data chunks to see the trade-offs between the data-transfer (communication) time and the kernel execution (computation) time on GPU and selecting the best match of the data chunk size for the target application that should be staged to the GPU. The need of transferring data to GPU comes from the fact that the input data is huge in biomedical applications and the GPU has limited on-chip memory and also, launching kernels on GPU with different execution configurations can't be accomplished without the intervention of CPU. However, the latter case has been addressed in newly Kepler-based GPUs by enabling dynamic parallelism where programmer can change the execution configuration of kernel in flight. This can help programmer porting more and more code to GPU without using general-purpose CPU to launch kernels onto GPU. Nonetheless, applications with large data-sets (much greater than GPU on-chip memory size) involves multiple stream with varying rates will happen by interleaving those streams into chunks and transferring these onto GPU for the required computations faithfully.

To cope with this issue given the huge amount of data of biomedical applications; sophisticated I/O techniques such as multiple buffering and pre-fetching can help which can overlap the data transfer from CPU to GPU with the computation on the GPU, and improve the performance by reducing the execution stalls (Venkatesan *et al*., 2010). This will not only keep the GPU busy, but will also keep on transferring the data in parallel to the computation on GPU such that chunk *N-1* is on fly onto GPU, chunk *N-2* is on its way from CPU to GPU and so on enabling double buffering. However, in Fermi-based setup, chunk *N* is on its way back to the CPU from GPU – empowering triple buffering. Such buffering can help in pipelining the process and hence, improving the overall throughput of the system.

From this survey, we conclude that the due to the massively parallel architecture of GPUs and the recent advances Kepler-based GPUs and even newer GPUs that will come in future will better leverage the biomedical algorithms along with the increase in the algorithmic complexity towards the increasing data demands of biomedical applications. This enables GPUs potentially viable in biomedical community to achieve the goal of real-time diagnosis and decision-making in clinical setup, and GPUs for bioinformatics will remain a vibrant research area in the future.

Novelty Statement

This is a review paper that discusses the trade-offs involved in the use of Graphics Processing Units (GPUs) for accelerating various steps in biomedical applications, especially those entailing bio-imaging. Such review work will help future researchers to leverage GPU intelligently and efficiently in order to find useful insights for disease diagnosis.

Author's Contribution

Abdul Hafeez: Main idea, state of the art discussions, future directions, drafted the paper
Akhtar Nawaz Khan: Writing/revisions and editing, provided research material
Zahid Ullah: Writing/revisions and editing, provided research material.

Conflict of interest
The authors have declared no conflict of interest.

References

Allec, S.I., Y. Sun, J. Sun, C.A. Chang and B.M. Wong. 2019. Heterogeneous CPU+ GPU-enabled simulations for DFTB molecular dynamics of large chemical and biological systems. J. Chem. Theory Comput. 15: 2807-2815.

Ben-Shalom, R., N.S. Athreya, C. Cross, H. Sanghevi, K.G. Kim, A. Ladd, A. Korngreen, K.E. Bouchard, and K.J. Bender. 2020.

NeuroGPU, soft-ware for NEURON modeling in GPU-based hardware. bioRxiv, 727560.

Bryer, A., J. Eric, F. Wright, M. Ferrato, T. Huber, E. Ortiz, R. Searles, S. Chandrasekaran and J.R. Perilla. GPU accelerated computation of isotropic chemical shifts offers new dimension of structure refinement in largescale molecular dynamics simulation. Biophys. J. 116: 569a-570a.

Cao, Y., D. Patnaik, *et al.*, 2010. Towards chip-on-chip neuroscience: Fast mining of neuronal spike streams using graphics hardware. Proceedings of the 7th ACM international conference on Computing frontiers, New York, NY, USA, ACM. https://doi.org/10.1145/1787275.1787277

Fessler, J.A.S.L., V.T. Olafsson, H.R. Shi and D.C. Noll. 2005. Toeplitz-based iterative image reconstruction for MRI with correction for magnetic field inhomogeneity. IEEE Trans. Signal Process., 53(9): 3393-3402. https://doi.org/10.1109/TSP.2005.853152

GraphStream, I., 2006. GraphStream scalable computing plateform (SCP).

Haldane, A. and R.M. Levy. 2021. Mi3-GPU: MCMC-based inverse Ising inference on GPUs for protein covariation analysis. Comp. Phys. Commun. 260: 107312.

Halder, J.D.H., S.-K. Song and Z.-P. Liang. 2008. Anatomically-constrained reconstruction from noisy data. Magnetic Resonance in Imaging. https://doi.org/10.1002/mrm.21536

He, X., S. Liu, T.-S. Lee, B. Ji, V.H. Man, D.M. York and J. Wang. 2020. Fast, accurate, and reliable protocols for routine calculations of protein–ligand binding affinities in drug design projects using AMBER GPU-TI with ff14SB/GAFF. ACS Omega, 5: 4611-4619.

Huang, T.Y., Y.W. Tang, *et al.*, 2011. Accelerating image registration of MRI by GPU-based parallel computation. Magnet. Resonance Imag., 5: 712–716. https://doi.org/10.1016/j.mri.2011.02.027

Hussong, R., B. Gregorius, *et al.*, 2009. Highly accelerated feature detection in proteomics data sets using modern graphics processing units. Bioinformatics, 25(15): 1937-1943. https://doi.org/10.1093/bioinformatics/btp294

John, D., M.H. Owens, D. Luebke, S. Green, J.E. Stone and J.C. Phillips. 2008. GPU computing. Proc. IEEE 96: 879-899. https://doi.org/10.1109/JPROC.2008.917757

John, E.S., D.J.H. Ivan, S. Ufimtsev and K. Schulten. 2010. GPU-accelerated molecular modeling coming of age. J. Mol. Graphics Model., 2: 116-125. https://doi.org/10.1016/j.jmgm.2010.06.010

Lewicki, M.S., 1998. A review of methods for spike sorting: the detection and classification of neural action potentials. Network Comput. Neural Syst., 4: 53-78. https://doi.org/10.1088/0954-898X_9_4_001

Lim, S., K. Kwon, *et al.*, 2009. GPU-based interactive visualization framework for ultrasound datasets. Comput. Animat. Virt. W., 20(1): 11-23. https://doi.org/10.1002/cav.279

Mathe, J., A. Aksimentiev, *et al.*, 2005. Orientation discrimination of single-stranded DNA inside the α-hemolysin membrane channel. Proc. Natl. Acad. Sci. USA, 35: 12377-12382. https://doi.org/10.1073/pnas.0502947102

NVIDIA, 2009. Whitepaper NVIDIA Next Generation CUDATM Compute Architecture: FermiTM.

NVIDIA, 2012. Whitepaper NVIDIA Kepler GK 110.

Pruessmann, K.P.M.W., P. Bornert and P. Boesiger. 2001. Advances in sensitivity encoding with arbitrary k-space trajectories. Magn. Res. Med., 46(4): 638-651. https://doi.org/10.1002/mrm.1241

Rafique, M.M., A.R. Butt, *et al.*, 2010. Designing accelerator-based distributed systems for high performance. 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid) IEEE. https://doi.org/10.1109/CCGRID.2010.109

Rafique, M.M., B. Rose, *et al.*, 2009. CellMR: A framework for supporting mapreduce on asymmetric cell-based clusters. Proceedings of the 2009 IEEE International Symposium on Parallel\and Distributed Processing, IEEE Computer Society: 1-12. https://doi.org/10.1109/IPDPS.2009.5161062

Romero-Ortega, M.I., A.R. Butt, *et al.*, 2009. Carbon nanotube coated high-throughput neurointerfaces in assistive environments. Proceedings of the 2nd International Conference on PErvasive Technologies Related to Assistive Environments, ACM. https://doi.org/10.1145/1579114.1579181

Sato, D., Y. Xie, *et al.*, 2009. Acceleration of cardiac tissue simulation with graphic processing units. Med. Biol. Eng. Comput., 9: 1011-1015. https://doi.org/10.1007/s11517-009-0514-4

Schiwietz, T., T. Chang, *et al.*, 2006. MR image reconstruction using the GPU. Proceedings of SPIE. https://doi.org/10.1117/12.652223

Schofield, D.J., L. Irving, L. Calo, A. Bogstedt, G. Rees, A. Nuccitelli and R. Narwal, 2019. Reclinical development of a high affinity α-synuclein antibody, MEDI1341, that can enter the brain, sequester extracellular α-synuclein and attenuate α-synuclein spreading in vivo. Neurobiol. Dis., 132: 104582.

Sehgal, A., N. Linduska, and C. Huynh. 2019. Cardiac adaptation in asphyxiated infants treated with therapeutic hypothermia. J. Neonat.-Perinat. Med., 12: 117-125.

Shen, W., D. Wei, *et al.*, 2009. GPU-based parallelization for computer simulation of electrocardiogram. Ninth IEEE International Conference on Computer and Information Technology, CIT IEEE. https://doi.org/10.1109/CIT.2009.134

Stone, S.S., J.P. Haldar, *et al.*, 2008. Accelerating advanced MRI reconstructions on GPUs. J. Parallel Distrib. Comput., 68(10): 1307-1318. https://doi.org/10.1016/j.jpdc.2008.05.013

Sumanaweera, T. and D. Liu. 2005. Medical image reconstruction with the FFT. GPU Gems, 2: 765-784.

Sutton, B.P.D.C.N. and J.A. Fessler. 2003. Fast iterative image reconstruction for MRI in the presence of field inhomogeneities. IEEE Trans. Med. Imag., 22(2): 178-188. https://doi.org/10.1109/TMI.2002.808360

Venkatesan, B.M., A.B. Shah, *et al.*, 2010. DNA Sensing using nanocrystalline surface-enhanced Al2O3 nanopore sensors. Adv. Funct. Mater., 20(8): 1266-1275. https://doi.org/10.1002/adfm.200902128

Vasconcellos, E.C., E.W.G. Clua, F.H. Fenton and M. Zamith. 2020. Accelerating simulations of cardiac electrical dynamics through a multi-GPU platform and an optimized data structure. Concurr. Comput.Pract. Exp. 32: e5528.

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

Zhang, J., J. Hills, Y. Zhong, B. Shirinzadeh, J. Smith, and C. Gu. 2018. GPU-accelerated finite element modeling of bio-heat conduction for simulation of thermal ablation. J. Mech. Med. Biol., 18: 1840012.

Zhuo, Y., X.L. Wu, *et al.*, 2011. Using GPUs to accelerate advanced MRI reconstruction with field inhomogeneity compensation. GPU Computing Gems, Emerald Edition. Elsevier. https://doi.org/10.1016/B978-0-12-384988-5.00044-9