An FPGA based Energy-Efficient Read Mapper with Parallel Filtering and in-situ Verification

Venkateshwarlu Yellaswamy Gudur, Sidharth Maheshwari, Amit Acharyya, *Member*, *IEEE* and Rishad Shafik, *Senior Member*, *IEEE*

Abstract—In the assembly pipeline of Whole Genome Sequencing (WGS), read mapping is a widely used method to re-assemble the genome. It employs approximate string matching and dynamic programming-based algorithms on a large volume of data and associated structures, making it a computationally intensive process. Currently, the state-of-the-art data centers for genome sequencing incur substantial setup and energy costs for maintaining hardware, data storage and cooling systems. To enable low-cost genomics, we propose an energy-efficient architectural methodology for read mapping using a single system-on-chip (SoC) platform. The proposed methodology is based on the q-gram lemma and designed using a novel architecture for filtering and verification. The filtering algorithm is designed using a parallel sorted q-gram lemma based method for the first time, and it is complemented by an in-situ verification routine using parallel Myers bit-vector algorithm. We have implemented our design on the Zynq Ultrascale+ XCZU9EG MPSoC platform. It is then extensively validated using real genomic data to demonstrate up to 7.8× energy reduction and up to 13.3× less resource utilization when compared with the state-of-the-art software and hardware approaches.

Index Terms—Field programmable gate array, genome sequencing, filtering, low-cost, low-energy, read mapping, verification

1 Introduction

The enormous advances in genomics in the past decade and the next-generation sequencing (NGS) methodologies have revolutionized the genomic research [1]–[3]. The information divulged by whole genome sequencing (WGS) has the potential to transform personalized treatment and diagnostics by the likes of early detection of health conditions, monitoring therapeutic progress and susceptibility to diseases, and thereby reducing morbidity and mortality [3], [4]. Genomic researchers are persistently investigating its role on predictive, preventive, participatory, and personalized (P4) medicine [2] with the aim of understanding the impact of genomic variants on 6000+ genetic diseases, including cancer. As such, it is reasonable to foresee that a significant fraction of population will have their genomes routinely analyzed at hospitals and clinics in the near future [1], [5], [6].

WGS is the process of determining the DNA sequence, i.e., the order of the nucleotide bases in the genome and its analysis. Sequencing and assembly pipelines of the WGS are responsible for obtaining the genomic data. The sequencing process identifies the structure of the genome as small randomly fractured segments, called reads, and the assembly pipeline re-assembles the actual genomes using reads. The process of assembling reads to construct the original genome using a reference genome as a blueprint is called read mapping. Owing to the Human Genome Project and enormous advances in NGS methodologies, the cost of sequencing is declining [7]–[9]. However, in read mapping, the resources for storing, processing, analyzing and other computational capabilities are not growing on par with the rate of sequence data generation [5], [8]. State-of-the-art methodologies in read mapping incur substantial infrastructure costs and are computationally intensive and also consume a vast amount of energy due to the volume of genomic data that are generated every day [5], [10].

In an effort to improve the performance of read mapping, researchers have traditionally resorted to accelerating using software algorithms and/or hardware implementations, implemented on CPUs, graphics processing units (GPUs), CPU+GPUs, field programmable gate arrays (FPGAs) [11]–[20]. However, they do not address the energy cost as a direct minimization objective, which needs a careful tradeoff between algorithmic design and implementation.

In this paper, we propose a novel energy-efficient read mapper using a single FPGA system-on-chip (SoC) platform. Read mapping includes two stages, filtering and verification [9], [14], [21]–[23]. The first stage identifies candidate locations in the reference genome for the reads and the second stage calculates the exact alignment of the reads to the reference genome. Core to our approach is a parallel filtering, complemented by *in-situ* verification for memory and computation compaction. The filtering algorithm is designed using a parallel sorted q-gram based method for the first time. Q-grams, derived from query reads, are used in the filtering operation by searching them in the reference genome using a binary search method, and the match results are used to determine the

- *V.Y. Gudur and A. Acharyya are with the Department of Electrical Engineering, Indian Institute of Technology Hyderabad, 502285, Telangana, India. E-mail: {ee15resch02009, amit_acharyya}@iith.ac.in.*
- *S. Maheshwari and R. Shafik are with the School of Engineering, Newcastle University, Newcastle Upon Tyne, UK, NE1 7RU. E-mail: {s.maheshwari2, Rishad.Shafik}@newcastle.ac.uk.*

candidate locations for the reads. Filtering is followed by an *in-situ* verification routine designed using parallel Myers bit-vector algorithm. For a given number of mismatches between query reads and the reference genome, the candidate locations where the reads may be located are examined in verification to align the reads to the reference genome. The proposed architectures for filtering and verification exploit the parallelism offered by FPGAs and are efficiently designed catering the need for low-energy consuming approaches of read mapping. The key ***contributions*** of this work are as follows:

- a q-gram lemma based novel methodology for parallel filtering;
- an *in-situ* verification using parallel Myers bit-vector algorithm for reducing memory footprint and increasing performance; and
- a single FPGA SoC based implementation featuring hardware/software codesign for energy and resource efficiency.

The proposed design is implemented on a Zynq Ultrascale+ MPSoC XCZU9EG FPGA platform. We compare our design with state-of-the-art software and hardware methods and show significant energy and performance-resource efficiencies. The average gain over energy consumption and resource utilization is 7.8× and 13.3× for software methods and 3.97× and 5.62× for hardware methods.

2 Related Work

The mapping of millions of reads obtained by sequencing process to a reference genome (~3.2 billion base-pairs in the human genome) is time and resource consuming. As such, heuristic approaches based on q-gram filters are used in read mapping [9], [22], [24]. These approaches of read mapping in genome sequencing include two stages, filtering and verification. For a given threshold of permissible number variations of the reads with respect to a section of the reference genome, the first stage eventually prunes the number of candidate locations for the reads by eliminating the parts of the reference genome where the chances of finding the reads are minimal [25], [26]. Using dynamic programming methods, the second stage compares and performs the alignment of the reads to the sections of the reference genome filtered by the first stage. Myers bit-vector algorithm is a popular algorithm for approximate string matching used for alignment in the verification stage [27]. It is dynamic programming based algorithm to find the edit distance between two strings using addition, shifting and other bitwise operations. It is extensively used in genomics for the problems of local and global alignment [28], [29].

Software based read mappers include BWA-MEM, Bowtie, mrFAST, SOAP, RazerS, SHRiMP2, GEM, RazerS 3, Yara, Hobbes3, CORAL and GRIM-Filter [28]–[37], [53]. In the filtration stage of many of these read mappers, pigeonhole principle and q-gram lemma approaches are used [9], [38]. In the pigeonhole principle, if a read is fragmented into $k+1$ segments, then in the approximate match of the read to the reference genome with k errors, there is at least one segment without error [39]. As short segments of the read are more likely found, the specificity of these filters is less. GEM mapper, RazerS 3, BitMapper, Hobbes3 and CORAL are the read mappers based on the pigeonhole principle [29], [34], [35], [37], [40]. A string of length q is called q-gram. In q-gram lemma, for a maximum of k errors, two strings each of length n share $n-(k+1)q+1$ number of common substrings [25]. In read mappers based on q-gram approach, a q-gram counting filter searches for regions in the reference genome that satisfies the condition for the minimum number of common substrings. The modified SWIFT, RazerS, SHRiMP2 and RazerS 3 are the read mappers based on q-gram lemma [26], [28], [29], [36]. Kim et al. present a processing-in-memory approach to perform filtering by exploring 3D-stacked DRAM memory technologies with high-memory bandwidth [53]. However, the portability of the hardware architecture is limited by their approach.

NGS methodologies generate a massive amount of sequence data, and it is doubling every 7 months [5], [8]. With growing performance demands, application-specific high-performance servers or clusters are the typical implementation choices using GPUs or FPGAs [11]–[17], [20]. In the literature, ASIC custom hardware accelerators are used for sequence alignment [54]–[56]. However, large monetary investments for chip design and the rigid architecture performing fixed operations limit the ASICs for wide adoption in genomics applications. Due to the high cost and energy consumption of GPU based clusters, FPGAs are often preferred for processing genomic data as they offer massive parallelism, low cost and high energy efficiency [12], [14], [18], [20]. Various algorithms for applications like pairwise sequence alignment, database searching, string matching, multiple sequence alignment, read mapping, etc., are designed using reconfigurable FPGA hardware [12], [18]–[20], [24], [41]–[44]. An extensive survey for sequence alignment using reconfigurable platforms is presented in [45].

A hybrid accelerator composed of both CPU and FPGA where only the seed generation and extension is implemented on FPGA is proposed in [44]. FPGA based MapReduce framework with multiple hardware accelerators is presented in [46] to align short reads to a reference genome. FM-index based approximate string matching for Bowtie running on FPGA using concurrent multiple hardware threads is presented in [47]. A multiple FPGA system for short read alignment with a maximum of two mismatches is presented in [15]. In this work, off-chip DRAM chips are used to store the expanded FM-indexes, and the time to reconfigure the FPGAs, transfer the FM-indexes and disk IO operations is omitted. An FPGA based system to accelerate and improve the NGS long read mapping using Smith-Waterman Algorithm is proposed in [48]. An FPGA based accelerated approach to compute edit-distance approximations for short read alignment is proposed in [17]. A hardware based pre-alignment filter using FPGA is proposed in [24]. Here, in addition to the memory bandwidth bottleneck, the standalone hardware filter requires existing read mappers to perform the verification.

To date, FPGAs are used as accelerators or co-

processors in separate filtering and verification pipelines within read mapping. To the best of our knowledge, a complete hardware system for the whole operation of read mapping using both filtering and verification is not available so far. In view of the above, we propose a low-energy read mapping methodology in Section 3. Initially, a system level description for the proposed methodology is presented. Then we explain our proposed q-gram lemma based parallel filtering and verification methodologies. In Section 4, the practical implementation of the proposed methodology is presented, together with its extensive validations. Finally, Section 5 concludes the paper.

3 Proposed Methodology for Read Mapping

Fig. 1 depicts the proposed read-mapping methodology. The methodology includes two levels, (a) preprocessing and (b) filtering and verification. Preprocessing is required for both filtering and verification. FPGA based hardware systems with limited memories are not useful for implementation of the whole operation of read mapping on the entire genome in one go. In order to facilitate working with memory restricted FPGAs and reduce the complexity of hardware required for filtering and verification operations, the reference genome is divided into smaller overlapping segments called sections. In the preprocessing for filtering, for every genome section, all the q-grams are obtained. These q-grams, which are in text format, are encoded into two-bit. The encoded q-grams have numerical values, and they are sorted in an array according to their values. For every genome section, there is a sorted q-grams array. As discussed later, this is helpful for a hardware compatible algorithm architecture design irrespective of the resources available on the hardware platform. In the preprocessing for verification, the sections of the genome are just two-bit encoded. Our methodology is generic and portable to any FPGA platform, irrespective of the hardware resources available on the platform.

A read is mapped to the reference genome within an edit distance, e, to accommodate sequencing errors and true variations in the original genome [33]–[35]. In our methodology for filtering, called *parallel sorted q-gram lemma based filter,* the filter identifies all those reads which can be mapped to the reference genome within the edit distance. Q-grams obtained from reads are parallelly matched with the q-grams in the sorted array of the genome section. The number of common substrings between the query read and the genome section derived for the edit distance, e, is compared with the total number of q-gram matches found in the array. The verification stage is selectively enabled whenever the number of matches crosses the number of common substrings. In the verification stage, the read is exactly aligned to the genome section using approximate string matching. The *in-situ* verification avoids memory-intensive operations that involve moving and saving the data in the filtering stage. Algorithm 1 gives an overview of the proposed methodology, including the above steps.

Fig. 2 gives the flow of preprocessing, filtering and verification. The details are as follows.

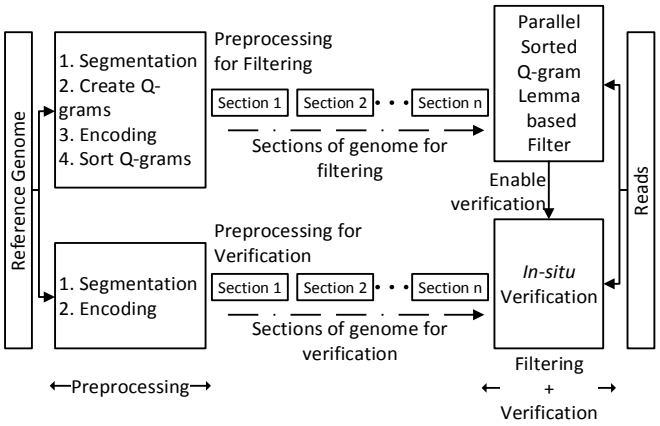

Fig. 1. System level representation of the proposed methodology.

A. *Preprocessing*

Fig. 2 (a) shows preprocessing for filtering, and it includes four steps, viz. segmentation, creation of q-grams, encoding and sorting. Initially, the reference genome is segmented into n small sections. For a read length R and edit distance e, the section size must be at least $R+e$ so that the given read can be mapped within the edit distance. A query read can be on the border of two sections. The overlapping part between the two sections has to consider mapping the read within the edit distance. As such, we choose $R+e$ as the overlapping size. Given a length of q-gram, q, all the possible q-grams are obtained from the section. From a section of length N, the number of q-grams X obtained from it is given as $X = N-q+1$. Nucleotides in the q-grams are in text format. To reduce the memory requirement, all the q-grams consisting of the nucleotides (A, C, G and T) are encoded in two-bit binary numbers (00, 01, 10, and 11). The N symbol is treated as any nucleotide, and one of the four nucleotides is randomly selected for every N symbol. All the encoded q-grams are sorted in ascending order according to their numerical values and stored in an array. The array is called a sorted q-gram array (SQA). For a given reference genome, sorting is required to be performed only a single time for creating the SQAs. In the proposed methodology, we do not limit the sorted q-gram array creation in the preprocessing step by choice of sorting algorithm. Any sorting algorithm can be used to obtain the SQAs. The bit vectors in [53] maintain a sorted order of q-grams similar to our approach. However, as discussed later, the two-bit

Algorithm 1. Flow of the proposed methodology for read mapping

Input: Reference genome, Reads
Output: Reads with their mappings
1 Preprocess the reference genome for filtering and verification
2 **for** each section of the genome
3 Filtering
4 **for** each read obtain all the q-grams
5 Search all the q-grams in the genome section for the filter
6 **if** total number of matches of all the q-grams exceed threshold
7 Enable verification for the read
8 Verification
9 **for** every read passed by the filter
10 Perform exact alignment for that read in the genome section

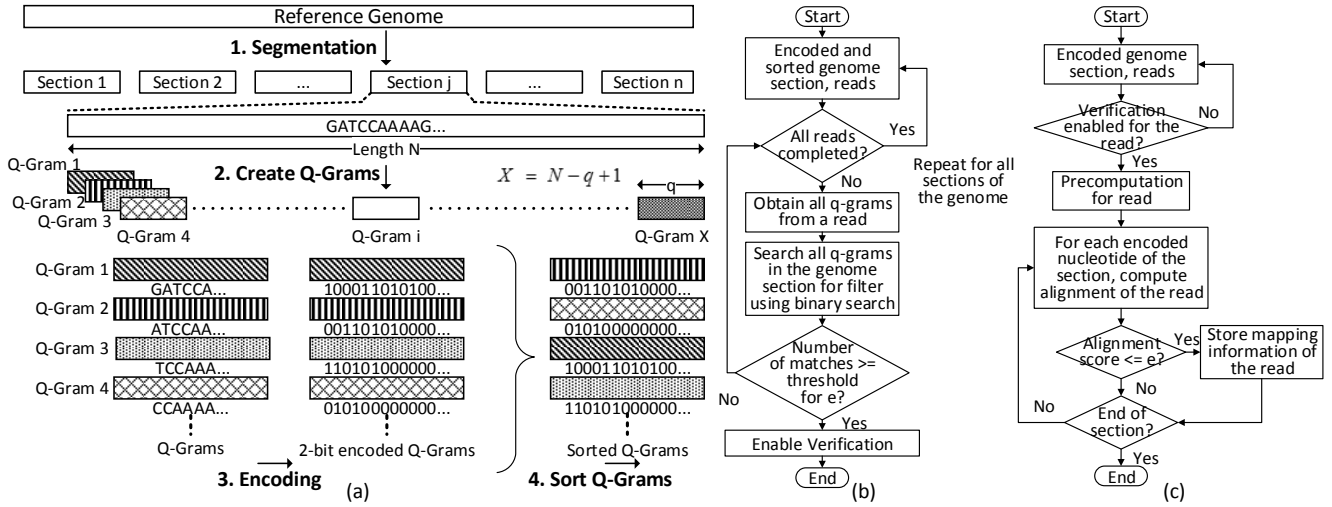

Fig. 2. Detailed steps in the proposed methodology (a) Illustration of preprocessing for filtering (b) Filtering flow (c) Verification flow

encoding of q-grams in our methodology facilitates binary searching in hardware and essentially reduces the memory transactions in the proposed hardware centric architecture. Preprocessing for verification includes two steps, segmentation and encoding. Each genome section from the n sections, is simply two-bit encoded, as explained earlier.

B. Filtering

For a practical value of e, usually 5% of the read length, it is observed that more than 98% of the reads exhibit incorrect mappings [21]. The computationally intensive exact alignment involving dynamic programming methods can be avoided by selectively enabling the verification. This is achieved by the filtering stage. Fig. 2 (b) shows the flow of filtering. The sorted q-gram arrays (SQAs) for all the sections are obtained from preprocessing. For every two-bit encoded read of length R, the number of q-grams M each of length q is given as $M = R - q + 1$. For a given number of mismatches or errors, the number of common substrings between the query read and the genome section is the threshold value required for enabling the verification. It is the minimum number of q-grams of the read required to match in the array. It is given as $C_{Theshold} = R - (e+1)q + 1$ where $C_{Theshold}$ is the count of q-gram matches, i.e., common substrings, R is the read length, e is the edit distance and q is the length of q-gram. As SQA involves sorted q-grams, binary search is efficient to search all query q-grams of the read [49]. Whenever the threshold value is met by the query read, a signal to enable verification is activated. For every genome section, the procedure is repeated for all the reads.

C. Verification

Fig. 2 (c) shows the flow of verification. In the verification stage, approximate string matching is employed to align the query read to the genome section. Genome sections are encoded in the preprocessing stage. The verification stage is designed using the Myers bit-vector algorithm [27]. Initially, in precomputation, two arrays are computed, one each for the bits of the nucleotides in the query read. It is followed by parallel bit-vector operations between the arrays and the two-bit encoded nucleotides of the genome section. The nucleotides from the section are passed sequentially. The alignment of the read with the section is quantitatively indicated by a score. The read is assumed to be mapped to a location whenever the corresponding alignment score for the location is equal to or lesser than the edit distance, e.

The detailed architecture of our method is presented in the next sub-sections. Initially, system-level architecture for read mapping using the proposed methodologies for filtering and verification is presented, followed by their detailed architectures.

3.1 System Architecture

The complete system architecture using the proposed methodologies for filtering and verification is given in Fig. 3. Initially, during the preprocessing phase, as shown in Fig. 1 and Fig. 2 (a), the reference genome is segmented into different overlapping small sections. For the filtering, this is followed by q-grams creation and two-bit encoding of the q-grams. The encoded q-grams of each section are sorted according to their binary values. For the verification, the segmented sections are two-bit encoded. Once the preprocessing is completed, the sorted q-grams for filtering and encoded sections for verification are stored in external memories. The reads obtained from a sequencing machine are also stored in the external memory.

The *System Control Logic* controls the filtering and verification and maintains the flow of the system operations. *Read FIFO*$_{In}$ is used to feed the read from the external memory to the system. The nucleotides of the reads are two-bit encoded, and transferring them to the system saves many memory operations. The reads are stored in a *FIFO* (first-in, first-out) memory inside the system. *Enable Filt Op* and *Enable Verif Op* are the global signals to enable the filtering and verification operations, respectively. The *Array* signals are used to update SQAs to the *Filtering* core. The *Section* signals are used to update encoded sections to the *Verification* core. *QSE En* is the binary vector used to enable or disable the q-gram search engines (QSEs) inside the filtering core, which is used to search a q-gram in the SQA. *Count Threshold* is the minimum number of common substrings used in filtering. *Score Threshold* is the edit distance, e, used to compare with the

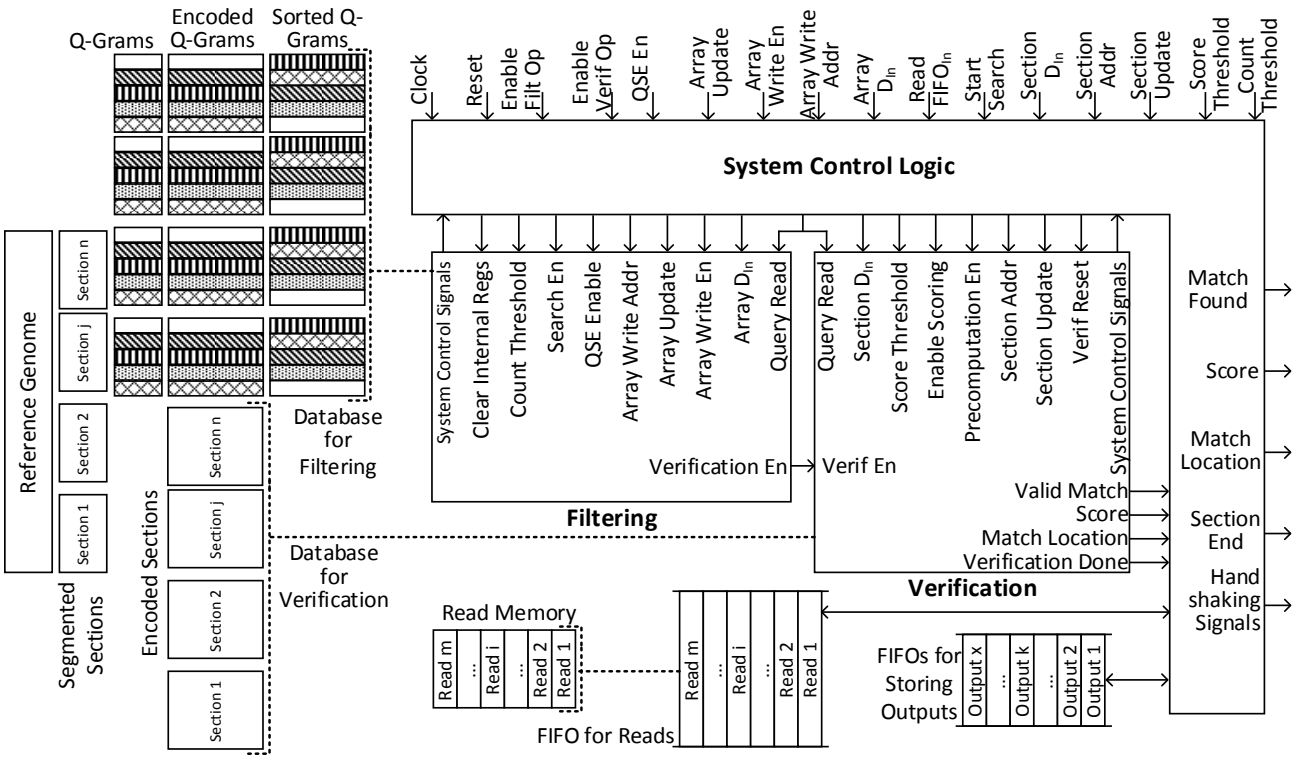

Fig. 3. Architecture of proposed methodology for read mapping using filtering and verification. Dotted lines indicate connection to memories.

alignment score in the verification. The mapping information of a read is indicated by *Match* signals and *Score*.

Initially, the SQA of a section of the reference genome is transferred to the filtering core using the *Array* signals. Then, the corresponding encoded section is transferred to the verification core using *Section* signals. The system begins mapping operation once the *Start Search* signal is activated. The encoded reads are sent to the system and stored in the *FIFO* memory from where they are inputted to both the *Filtering* and *Verification* cores. Filtering is performed on the reads, and depending on the *Verification En* signal, the *Verification* core activates the read and decides to perform verification. On activating the *Verif En* signal by the *Filtering* core, the *Verification* core aligns the read to the section of the reference genome and completes the mapping of the read. The outputs related to the mapped read are stored in *Output FIFOs*. The outputs can be combined and stored in a single *FIFO,* or multiple *FIFOs* can be used for each output signal. *Handshaking Signals* are used at various stages for multiple purposes, e.g., to indicate the start and end of transferring operations like transfer of genome sections and reads, to indicate the end of searching of all reads, to communicate with the top-level modules, and to interact with internal modules using the system control signals. Once the mapping of reads for the section is completed, on the next section, above operations are performed in a similar fashion.

Large memory requirements inside the system and the associated memory intensive operations for storing the entire reference genome are big drawbacks in FPGA based read mapping methods. In the proposed methodology, as read mapping is performed by dividing the reference genome into smaller sections, these drawbacks are eliminated. Depending on the availability of the system resources, the size of the sections can be determined. However, as experimentally validated later, smaller sizes of sections eliminate most of the incorrect mappings and thereby reduce the computational burden on verification. Such a choice of small section size increases the filtering time, as all the reads are repeatedly searched in the sections. Choosing larger sizes of sections leads to filtering of many reads as the likelihood of q-grams matching increases and the threshold of common substrings is met. This, in turn, increases the time and computational burden on the verification stage. Optimum section size has to be decided so that most incorrect mappings are eliminated while simultaneously the time required for filtering and verification operations is maintained.

3.2 Filtering using Parallel Q-grams

In this sub-section, we present the q-gram search engine for searching q-grams in a section of the genome. It is followed by the q-gram based filtering methodology that uses multiple q-gram search engines in parallel for searching q-grams of a read in a section of the reference genome.

3.2.1 Q-gram search engine architecture

Q-gram search engine (QSE) is the core used to search q-grams obtained from a read in the SQAs using binary search. Fig. 4 shows the architecture for the QSE using the concept of sorted q-grams. The *QSE En* signal is used to enable and disable the search engine core. The local memory is updated with SQAs using the *Array* signals. *Search En* signal is used to indicate the start of the searching operation for a query q-gram in the array. For a two-bit encoded *Query Q-Gram* and the corresponding binary value, the control logic searches for the q-gram in the lo-

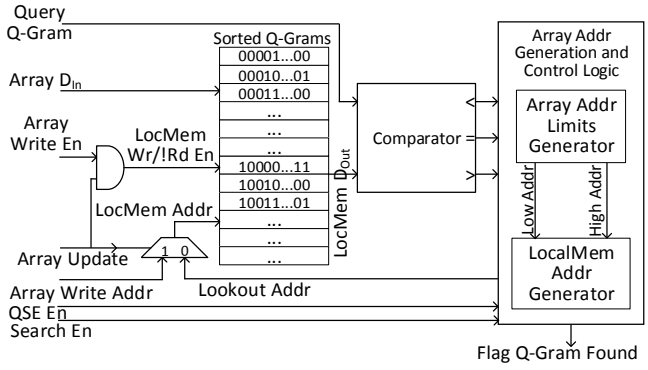

Fig. 4. Detailed hardware architecture of q-gram search engine.

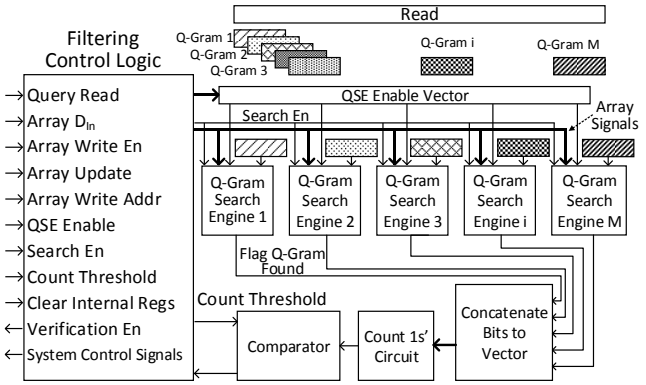

Fig. 5. Detailed hardware architecture of the proposed parallel filtering methodology using sorted q-grams.

cal memory array. The local memory contains an SQA. As such, binary searching is an efficient method, and it is implemented to find the q-gram in the SQA [49]. A comparator compares the binary value of query q-gram with the sorted q-gram produced for a corresponding address. Depending on the comparator output, the *Array Address Limits Generator* circuit computes the low-end and high-end address values required in the binary search algorithm for calculating the next address of the local memory with the help of *Local Memory Address Generator*. The *Array Update* signal is used to connect the address lines of the local memory to the write and read addresses during the updating of memory and searching operations, respectively. The control logic outputs a flag whenever the query q-gram is found in the local memory containing the SQA. The searching operation takes a maximum of $\log_2(X)$ clock cycles where X is the number of q-grams obtained from the corresponding section of the reference genome.

3.2.2 Proposed filtering using q-gram search engines

The proposed filtering methodology requires searching all the q-grams obtained from a read. To search M q-grams in parallel, M QSE cores are required. Fig. 5 shows the architecture for the filtering using QSE cores in parallel. Each core in the architecture can search for a q-gram in the SQA representing the genome section. All the M q-grams are searched in parallel and in turn, the entire read is filtered in the section. There are as many numbers of search engine cores in the architecture as the number of q-grams of the read. The *Array Signals* are composed of signals required to update the QSE cores with SQAs obtained from the sections of the reference genome. A multi-bit signal, *QSE Enable Vector*, is generated by the *Filtering Control Logic* for selective enabling and disabling of the QSE cores using the value of the *QSE Enable*. The binary value of *QSE Signal* corresponds to the particular QSE number to be enabled/disabled. Special values like all 1's or 0's can be used to turn on/off all the QSE cores. Each two-bit encoded q-gram obtained from the *Query Read* is searched by the QSE core in its local memory containing the SQA of the genome section. *Count Threshold* is the minimum number of q-grams required to match in the SQA. The *System Control Signals* are used for miscellaneous activities and handshaking between the filtering core and other modules of the system.

The *Search En* signal enables the searching of query q-grams of the read. All the QSE cores work in parallel and search their respective q-grams in the SQAs stored in the local memories. The entire query read comprising of M q-grams are searched in just $\log_2(X)$ clock cycles, which is required by a single QSE engine for a section from which X q-grams are obtained. Whenever a q-gram is identified in the local memory, the QSE core outputs a flag to indicate the match. The *Count 1s' Circuit* sums all the flags obtained from the QSE cores. The *Comparator* compares the sum with the value of *Count Threshold*. Whenever the required number of q-grams are matched in the array (i.e., section of the reference genome), the *Verification En* is activated to enable the verification and alignment of the query read in the section of the reference genome. The accurate alignment of the query read in the section is accomplished by the verification core as described later. If the number of q-grams matched in the section is less than the *Count Threshold* value, it indicates that the section of the reference genome is not a candidate location for the query read. It is an incorrect mapping for the query read, and thus the section of the reference genome is rejected for verification of the corresponding query read. This selective enabling of the verification by the filtering core assists in reducing the load on the computationally intensive verification and alignment operation and thereby conserving time and resources.

3.3 Memory-aware in-situ Verification

A genome section is a candidate location for every read with common substrings more than the threshold value. In the proposed verification, the read is exactly aligned with the genome section using hardware based Myers bit-vector algorithm [27]. In the original Myers bit-vector algorithm [27], CPU registers are directly used for reads up to 64 base-pairs. Reads of 65 base-pairs and higher require multiple CPU registers to emulate the bit-vectors and operations [27], [29]. However, there is performance degradation due to additional processing overhead caused by operations on multiple registers. The banded version of the Myers bit-vector algorithm is four times faster than the original algorithm [57]. However, it uses multiple conditional loops and controlling statements leading to a complex state machine and architecture when implemented on hardware. As a result, there might be limitations on the frequency of the operations of the hardware that leads to degraded speed performance. In

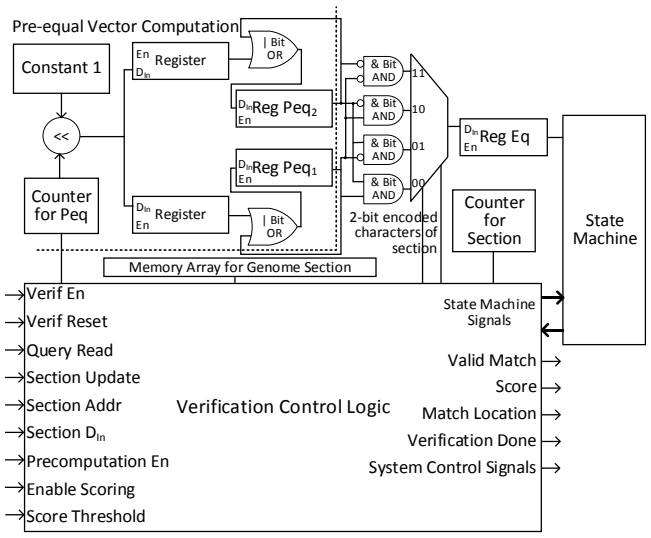

Fig. 6. Detailed hardware architecture for verification

the proposed Myers bit-vector algorithm based verification using hardware processing on an FPGA, there is no limitation on reads size as the entire read can be processed in a single iteration without the need for multiple loops. In addition, to design a simple hardware architecture supporting a maximum operating frequency without timing violations, the original Myers bit-vector algorithm is used instead of its banded version.

Fig. 6 shows the architecture of the proposed verification methodology. The encoded genome sections, in which alignment has to be done, are transferred to the verification core and stored in the *Memory Array* using the *Section* signals. In the filtering stage, the *Verification Enable* signal is activated for a *Query Read* whenever the number of q-grams matched in the genome section is higher than the *Count Threshold*. This signal, indicated by *Verif En*, is used to enable the verification core.

In the precomputation stage, the pre-equal vector is computed for the given read using the top region of the architecture. As the read is two-bit encoded, the pre-equal vector is divided into two vectors, one for each bit of the nucleotide. When the *Precomputation En* signal is enabled, the pre-equal vector, comprised of *Peq2* and *Peq1,* is calculated for each nucleotide of the read. *Counter for Peq* is used to keep track of the nucleotide position in the read. The bits of *Peq2* and *Peq1* vectors corresponding to the location of the nucleotide in the read are set to the two-bit encoded value of the nucleotide. This is achieved as follows: a constant 1 is left-shifted by the count value and the *Peq* registers enabled by the *Verification Control Logic* are bitwise *ORed* with the shifted value. Once the computation of the pre-equal vector is completed, *Enable Scoring* signal is activated.

Operations between the bits of *Peq* registers and the two-bit encoded nucleotides of the genome section occur in parallel. For each nucleotide of the genome section, the *State Machine* compares the nucleotide with A, C, G, and T and computes the score of alignment. The input line of the multiplexer corresponding to the current two-bit encoded nucleotide of the section is activated and connected to the register of the equal vector, indicated by *Reg Eq*. The state machine runs through multiple states to compute different bit-vectors, as described in the Myers bit-vector algorithm, and performs operations parallelly on all the bits of the read using the bit-vectors. A separate counter, *Counter for Section,* is used to keep track of the nucleotides of the genome section. For the sake of simplicity, the enable signals from the control logic to the registers and counters are not connected. The edit distance value, e, is indicated by *Score Threshold*. It is the score required to indicate a valid mapping for the read. The read alignment to the genome section within the permissible edit distance is indicated by the *Valid Match* signal. *Verification Done* is used to indicate the end of the verification operation, and it is also used to resume the filtering whenever it is halted for verification. For every valid mapping, the corresponding *Score* and the *Match Location* of the read in the genome section can be read in the top-level system where the verification core is instantiated.

As described in the earlier sections, the verification core is placed alongside the filtering core. This in situ verification of reads in the sections of the reference genome is helpful to realize a complete system on a single FPGA device. The filtering output generated for all the reads in all the sections is large and storing it is memory intensive. Verification is performed for all the eligible reads as indicated by the filtering. As such, there is no need to store the filtering output. Thus, the need for intensive memory operations related to the data of filtering is eliminated by the proposed *in-situ* verification, thereby achieving a compact and memory-aware implementation.

4 Evaluation

In this section, the implementation of the proposed methodology is presented, followed by a detailed evaluation and comparison with the existing literature.

4.1 System Implementation on SoC FPGA

The system designed using the proposed filtering and verification methodologies is implemented on a system-on-chip (SoC) FPGA device. SoC FPGAs have a processor to run the software and a reprogrammable logic to design any user specific hardware [50]. Hardware/software codesign incorporating the advantages of both the hardware and software is used to implement the system [20], [51]. Fig. 7 presents the system-level implementation for read mapping. Many of the control actions required by the system, including the transfer of sections and reads, controlling of outputs, and handling of external memories, necessitates a sophisticated control and state machine. In such a scenario of many control actions, an intelligent processor comes in handy. The *Processing System* (PS) may include any softcore processor or an ARM processor on SoC FPGAs and executes the software part of the system. The ZCU102 evaluation kit from Xilinx designed using the Zynq UltraScale+ MPSoC FPGA and Xilinx Vivado Design Suite 2017.3 software tools are used for the implementation following the embedded system design flow [50]. The FPGA device on ZCU102 kit has

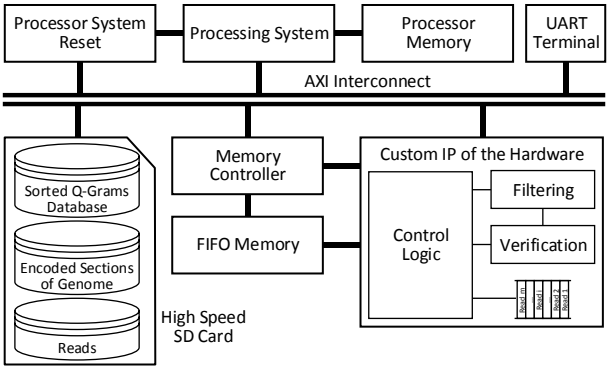

Fig. 7. System level implementation of the proposed methodology.

ARM Cortex-A53 processor, and it can be used as the processing system to run the flow of the system as described in Algorithm 1. In addition to the PS, the FPGA device also includes reconfigurable hardware fabric required for the implementation of the hardware system shown in Fig. 3. C program is written to run on the PS to perform all the control actions, and it acts as the top-level system where the read mapping system shown in Fig. 3 is called using hardware drivers.

The *Processor System Reset* is used to provide a synchronous reset to the entire system, including the hardware blocks. The *Processor Memory* is used as system memory for the PS, and it is made of on-chip or on-board DDR memory. As on-chip memory is limited for FPGA devices, on-board DDR memory is suitable for programs that create a large memory footprint. The *Memory Controller* facilitates communication between memory modules and PS. It is also useful for sending and receiving high-speed data from custom designed blocks, for example, the hardware system. The sorted q-grams and encoded genome sections and reads are stored on external memories, including USB drive, SDHC card, and SATA hard disk. In our implementation, we use high speed Secure Digital High Capacity (SDHC) card. The SD card is interfaced to the PS using Secure Digital Input/Output (SDIO) interface. The *UART Terminal* can be used to display messages indicating the status of the system and also to interact with the user. If selected by the user, it may also be used to show the mapping results of the system. All the peripherals are connected to the PS using *Advanced Extensible Interface* (*AXI*) *Interconnect*. The entire system shown in Fig. 3 is packaged as an IP and interfaced to the PS. It is indicated by *Custom IP of the hardware*. The *Custom IP* is instantiated in a top module, including all the blocks of Fig. 7. The outputs obtained from the *Custom IP* are stored in the *FIFO Memory*. The output is read by the PS and depending on the user's selection, either it can be displayed on *UART* or stored in a separate file on the SDHC card for further studies.

The entire flow of the system implementation and its working is presented in Algorithm 2. Initially, the user shall be asked whether the preprocessing is already performed or need to perform freshly on the reference genome stored on the SDHC card. In case preprocessing is performed by an external computer, the files related to the sorted q-grams of filtering and the encoded sections of verification are stored on the SDHC card. For a fresh pre-

Algorithm 2. Flow of the system implementation using the proposed methodology for read mapping

Input: Reference genome **RG**, Reads **Rds**
Output: Reads with their mappings **Op**
1 If preprocessing files are present proceed to 3
2 Perform preprocessing and write sorted q-grams and encoded section files on SD card
3 Read files: sorted q-grams into fileSoQ, encoded sections into fileSeE and reads into fileRds
4 **while** fileSoQ and fileSeE not empty **do**
5 Read section from fileSoQ and send to filtering, read section from fileSeE and send to verification and start search
6 **while** fileRds not empto **do**
7 Read the reads from fileRds and send to Custom IP
8 Obtain output and store in Op on SD card or display on UART
9 **end**
10 **end**

processing by the system, original text file of the reference genome is expected on the SD card and the output files from preprocessing are stored on the same card. Once the preprocessing files are ready, system control reads the sorted q-grams and encoded sections from their corresponding files on the SD card. The system control identifies and distinguishes between the sections in the files as indicated by special characters and transfers them to the filtering and verification cores through the interface to the *Custom IP*. After sending the sections, the reads are obtained from their corresponding file. The reads are two-bit encoded on the fly and sent to the *FIFO Memory* using the memory controller. The encoding time of the reads doesn't affect the system performance as the conversion is done in software in parallel to the operations in *Custom IP* and FIFO type memory is used for storing reads. The filtering and verification operations are performed on the reads as described in earlier sections. The outputs obtained from the *Custom IP* are stored in the output *FIFO Memory* and read by the PS. The output is either displayed on *UART* or stored in a separate file on the SD card. The flow of the entire system is maintained by the software as all the control actions are performed by the C program running on the PS.

4.2 Experimental Results and Analysis

In this sub-section, we evaluate the performance of the proposed low-energy and low-cost methodology for read mapping. Initially, resource consumption for the implementation of the proposed architectures is presented. It is followed by the effect of different section sizes on mapping time. Later, the sensitivity and accuracy results of the proposed methodology as compared to a gold standard in the literature are presented. Finally, a detailed comparison with state-of-the-art software and hardware methodologies in the literature is presented.

4.2.1 Resource utilization

The proposed read mapping methodology presented in Fig. 3 and implemented in Fig. 7 are targeted for Zynq UltraScale+ XCZU9EG-2FFVB1156 device and implemented using Xilinx ZCU102 evaluation kit and Vivado 2017.3 Design Suite software tools. To evaluate the optimum value for section size as stated in previous section, the architecture is implemented for different section sizes. The architecture, including sections of 512, 1024, 2048,

TABLE 1
Resource Utilization on FPGA Implementation for Different Section Sizes

Section Size	512				1024				2048				4096				8192			
Resource	LUT	LMem	FF	BRAM	LUT	LMem	FF	BRAM	LUT	LMem	FF	BRAM	LUT	LMem	FF	BRAM	LUT	LMem	FF	BRAM
QSE	126	0	47	0.5	131	0	52	1	137	0	57	2	151	0	62	4	167	0	67	7.5
Filtering	11014	0	4038	42.5	11352	0	4467	85	11834	0	4896	170	12913	0	5325	340	14376	0	5754	637.5
Verification	1531	0	1653	0	3327	0	2683	0	5417	0	4735	0	12623	0	8835	0	23660	0	17028	0
System	18730	203	14448	882.5	30656	8907	16651	912	82609	54475	20393	912	184950	143563	25774	912	138766	86219	32077	906

**QSE: Q-gram search engine; resource type on FPGA, LMem: LUT Memory, FF: Flip-flop, BRAM: Block RAM. All the implementations are run using Vivado 2017.3 Design Tools and targeted on Xilinx XCZU9EG-2FFVB1156 FPGA device.*

4096 and 8192 sizes, is designed, and the corresponding post-implementation resource utilization for a clock frequency of 187.498 MHz is given in Table 1. Here, all the architectures are designed to map reads of 100 base-pairs. Long reads require a large number of parallel q-gram search engines leading to more requirement of on-chip memory. However, the filtering architecture can be modified to work in batches of q-grams where each batch contains a maximum number of q-grams as decided by the available memory resources. The maximum available resources on the FPGA are 274,080 (LUT), 144,000 (LUTRAM), 548,160 (Flip-flop) and 912 (BRAM). It can be seen that resource utilization is rising proportionately with section size. When the complete system is implemented, additional BRAM resources are utilized for buffering the query reads and also for implementing FIFO memories. For each section size, the corresponding power consumption for the entire system is 4.404, 4.435, 4.538, 4.753 and 4.737 watts, respectively. Out of these total power consumption values, the ARM core (PS) consumes 3.2 watt. The sorted q-gram file of chromosome 20 is ~688 MB, while the encoded section file is 44 MB. For 3.2 billion base-pair, corresponding files are ~34 GB and ~2.2 GB, respectively. The 64 GB SDHC card interfaced with the ZCU102 kit can conveniently store the files of a sufficiently large reference genome.

4.2.2 Effect of section size on read mapping time

The effect of section size on filtering and verification time is studied in this sub-section. In the experiments, all the studies are performed using real reads and genomes downloaded from EMBL-ENA (www.ebi.ac.uk/ena) and UCSC Genome Browser. The read simulator software *mason* (mason2-2.0.9) [52] is used to generate simulated reads of 100 base-pairs with different errors. These reads are mapped to chromosome 20. The time required to send the reference sections from the SDHC card and write the outputs is negligible compared to the filtering and verification time. The proposed architecture for read mapping supports working on different sections and reads in a streaming fashion without affecting the overall mapping time. As such, we only consider filtering and verification time to emphasize the proposed methodology and architectures. The total time for read mapping, including filtering and verification, is given in Fig. 8. Here, 6 sets of 100,000 reads are used for mapping, and average values obtained for a frequency of 187.498 MHz are presented in the chart.

Initially, at lower values of edit distances, for increasing section size, it is observed that the total mapping time reduces as the corresponding filtering time reduces with section size. Filtering time is approximately halved for increasing section size. As the number of errors allowed for lower edit distance values is small, the value of corresponding verification time is also small because the number of reads passed by the filter is very few. Later, for higher values of edit distances, it is observed that the likelihood of q-gram match increases and the number of reads identified by the filter for verifying in the candidate locations also increases. As such, for increasing edit distance values, verification time dominates the entire mapping operation. In the evaluation of the total time for different section sizes, it is evident from the chart that the architectures with sections of 2048 and 4096 sizes perform better as compared to other section sizes. In this study, section sizes of 128 and 256 are not considered as only the filtering time alone is observed to be more than 6.3 and

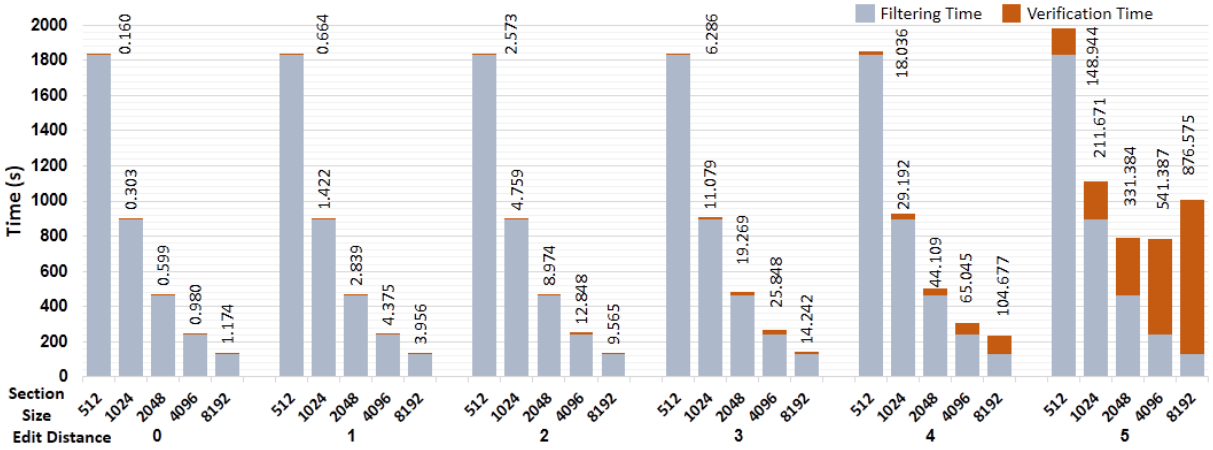

Fig. 8. Effect of section size on read mapping for different edit distances. 6 read-sets are used. The proposed method is designed using 2048 section size running at a frequency of 187.498 MHz. In the chart, only verification time is labelled in seconds.

TABLE 2
Number of Locations Reported by the Proposed Methodology

Type of Reads	Chromosome	Edit Distance					
		0	1	2	3	4	5
Simulated	2	4214	17697	39368	62237	80690	93673
	8	4175	17387	39055	61840	80641	93584
	20	4200	17768	39378	62300	81149	94229
Real	2	3931	5709	7601	9855	12485	15718
	8	2491	3919	5608	7743	10204	13147
	20	1312	2398	3943	6086	8971	12567

**Two types of reads each with 3 sets of 100,000 reads obtained from ERR240727_1 and mason software with 100 base-pairs size are mapped to different chromosomes. Accuracy of the reported locations and edit distances is on a par with the gold standard RazerS 3.*

1.6 hours, respectively. In order to strike a balance between resource utilization and the total time taken for mapping, 2048 can be safely chosen as an optimum value of the section size. In the subsequent sections and experimental evaluations, the architecture with 2048 section size is used in all the studies.

4.2.3 Evaluation of mapping sensitivity

The mapping accuracy of the proposed methodology is studied by mapping real and simulated reads. Real reads are obtained from the ERR240727_1 reads-set of 100 base-pairs size. *Mason* (mason2-2.0.9) [52] software is used to produce simulated reads of 100 base-pairs size from the chromosomes. Illumina profile is used with all the error model values set to 0.01. For each chromosome, 100,000 reads of both the types are mapped using the system designed with the 2048 section size. RazerS 3 is an all-mapper and has the highest accuracy and sensitivity compared to other methods [29]. In the literature, it is used to build the gold standard for Rabema [58] accuracy benchmarking [33], [35], [37]. As such, due to its accuracy, sensitivity, and all-mapper capability, RazerS 3 [29] is chosen as a gold standard in the evaluation of mapping sensitivity of the proposed methodology.

In the proposed methodology, we do not produce the CIGAR string and SAM output format yet. However, we compare RazerS 3 and the proposed methodology outputs in terms of read identification, matched location and edit distance (score) with include *all* accuracy of Rabema benchmarking [58]. The number of locations obtained for all the reads mapped with different edit distances is reported in Table 2. When edit distance values ranging from 0 to a maximum of 5 (5% of 100, [21]) are considered, it is found that the number of mappings given by RazerS 3 and the proposed methodology is equal. Also, when outputs of both the methods are parsed, it is observed that the mapping locations and edit-distance values are equal. However, a few mapping locations are observed to be slightly varying within a range of the corresponding edit distance value. As indicated in [29], there can be multiple beginning positions of a read and these variations in mapping locations of reads are expected while working with a dynamic programming based alignment algorithm. In evaluating the accuracy and sensitivity score for different edit distance values, the proposed methodology is found to have a 100% score. By this study, it is experimentally validated that the proposed methodology is fully sensitive and accurately generates the mapping information. The proposed read mapping methodology has the all-mapper capability and is considered as an *all-mapper* method.

4.2.4 Effect of reference size

In the previous sub-sections of evaluation, chromosome 20 is used as a reference for mapping. In this sub-section, different chromosomes and the effect of their sizes on mapping-time are studied for varying edit distances. The selected chromosomes are 2, 8 and 20 each of 242,193,528, 145,138,636 and 64,444,167 nucleotides. For a clear understanding of the effect of edit distance, only simulated reads are used in this study. For each chromosome, 6 sets of 100,000 reads are produced using *mason* software (18 sets in total). The average values of mapping time for edit distance ranging from 0 to 5 are given in Table 3. In all the experiments on a chromosome, the filtering time is nearly constant for different values of edit distance, whereas verification time increases with edit distance. The increase in verification time is maintained, as explained in earlier sections that more reads are needed to be verified and aligned for higher values of edit distance. It is observed that the filtering time is proportional to the size of the reference chromosome under consideration, and thereby the reference size directly influences the total time.

4.3 Performance Evaluation

A detailed comparative study and analysis with the existing software and hardware methods for read mapping are presented in this section. Initially, the proposed methodology is compared with state-of-the-art software methods followed by hardware methods. In order to justify the comparison with a few hardware-based sequence alignments, the proposed methodology is also evaluated with the sequence alignment operation.

4.3.1 Efficiency in comparison with software methods

TABLE 3
Effect of Reference Size on Filtering and Verification Time

Edit Distance	Chromosome 2			Chromosome 8			Chromosome 20		
	Filtering	Verification	Total	Filtering	Verification	Total	Filtering	Verification	Total
0	1731.763	0.290	1732.053	1037.790	0.274	1038.065	460.796	0.599	461.395
1	1731.788	2.303	1734.091	1037.790	1.558	1039.348	460.796	2.839	463.635
2	1731.775	7.521	1739.296	1037.787	5.251	1043.038	460.794	8.974	469.769
3	1731.730	16.412	1748.142	1037.773	11.644	1049.417	460.786	19.269	480.055
4	1731.668	39.003	1770.671	1037.732	28.688	1066.420	460.757	44.109	504.866
5	1730.980	612.847	2343.827	1037.323	345.255	1382.578	460.465	331.384	791.849

**All the time values are in seconds. Size of the chromosomes 2, 8 and 20 are 242,193,528, 145,138,636 and 64,444,167 bases. Reads are simulated using the mason software and multiple read-sets (total 18 sets, 6 for each chromosome) are used.*

The proposed methodology is compared with state-of-the-art software methods available in the literature. In this comparative study, multiple sets of 100,000 reads of 100 base-pairs size are mapped to chromosome 2, 8 and 20 with an edit distance of 3 as some software allows only a maximum value of 3. For each chromosome, 6 real and 6 simulated read-sets are considered. The results obtained are averaged for these multiple sets. All the software methods are run on a computer system with Xeon E5-2650 v2 CPU @ 2.60 GHz with 32 GB RAM and running Ubuntu Linux (version 18.04.2) operating system. A power meter is connected between the mains power supply and the computer system, and average values of power are tabulated in the table. All the software methods are invoked by their respective commands and indexing is performed wherever required. While performing the mapping, all unwanted processes are turned off, and only the basic processes essential for the software are run.

In order to have a fair comparison between the software running on CPU and the proposed methodology, a single thread is used to run the software (wherever applicable), and a single FPGA is used to run the proposed methodology. The total energy consumption for the read mapping is dependent on the application power. As it is dependent on the application and its run time, it is called dynamic power. It is the difference between the average power values observed while the software is running and the idle system, i.e., no software is running and the system is in idle state. Table 4 shows the average power values and mapping time, while Table 5 shows the average energy consumption and energy efficiency for normal and dynamic cases. It is evident from Table 5 that the proposed methodology requires the least amount of energy to perform the mapping. The results obtained in dynamic conditions assist better in comprehending the energy consumption associated with the read mapping and establish a fair comparison of the software methods and the proposed method. On average, it is observed that per unit energy, the proposed methodology is 7.8× more energy efficient than the software methods while mapping the reads. In comparison with application energy, i.e., dynamic conditions, efficiency is observed to be 12.12×. For chromosomes 2 and 8, though the best software method Hobbes3 is observed to be less energy consuming than the proposed methodology, in comparison with the application energy, the proposed methodology is more efficient. In addition to the gain over energy consumption, the proposed methodology is entirely implemented on a single SoC FPGA hardware platform avoiding the use of costly infrastructure.

TABLE 4
Power and Time Performance of the Proposed Methodology in Comparison with Software Methods

Method	Version	Pavg (W)	Pavg-Dy (W)	Mapping Time (s)		
				Chr 2	Chr 8	Chr 20
Bowtie2 [31]	2.3.5.1	75.62	21.607	3969	1949	1315
BWA-MEM [30]	0.7.17	77.04	23.028	128.89	95.41	71.13
GEM [34]	3.6.1.	75.62	21.607	235	211	157
Hobbes3 [35]	3.0	78.39	24.379	71.47	56.55	31.24
mrFAST [32]	2.6.1.0	76.97	22.957	1139.97	626.05	604
RazerS [28]	1.5.8	75.04	21.028	230.23	138.76	95.19
RazerS3-P [29]	3.5.8	76.81	22.799	136.21	87.99	63.79
RazerS3-S [29]	3.5.8	75.81	21.799	175.74	108.06	78.17
Yara [33]	0.9.11	75.62	21.607	542.01	342.56	204.48
Proposed	-	4.57	0.851	1748.14	1049.41	480.06

** A CPU with Xeon E5-2650 v2 @ 2.60 GHz with 32 GB RAM running Ubuntu Linux (version 18.04.2) operating system is used to run all the software. In this study, 6 sets of real reads obtained from ERR240727_1 and 6 sets produced using mason software are used for each chromosome. Average values of power and time are reported. In RazerS3, -P and -S stands for pigeonhole and swift filters. Dy– is used to indicate measurements observed during dynamic power consumption. Chr– chromosome.*

In order to normalize and study the energy and cost efficiencies, we define two metrics PCpJ and PCpsWD. The first metric is the number of pairs compared (PC) per unit energy, where PC is obtained by the product of the number of reads, read size and reference size (PC/J). The second metric is a performance metric per all the required resources, including time, power and cost. It is given as throughput per unit watt per $ (PC/s/W/$). The cost of the FPGA used for implementation is $2,495, while the cost of the computer system used to run all the software methods is $4,250. The normalized energy and resource efficiencies, along with the dynamic results, are plotted in Fig. 9. In comparison with Hobbes3, as observed to be the best among all the software methods, in dynamic conditions, the proposed methodology is observed to be 1.17–1.86× more energy-efficient. In terms of performance-resource efficiency, the gains in normal and dynamic conditions are 1.19–1.89× and 1.99–3.17×. In comparison with all other software methods, the gains over energy and resources are 1.3–45.3× and 2.1–77.1× in normal and

TABLE 5
Energy Consumption and Efficiency of the Proposed Methodology in Comparison with Software Methods

Method	Energy (J)			Energy-Dy (J)			Energy Efficiency (Reads/J)			Energy Efficiency-Dy (Reads/J)		
	Chr 2	Chr 8	Chr 20	Chr 2	Chr 8	Chr 20	Chr 2	Chr 8	Chr 20	Chr 2	Chr 8	Chr 20
Bowtie2 [31]	300147.2	147389.0	99444.1	85756.4	42111.1	28412.6	0.333	0.678	1.006	1.166	2.375	3.520
BWA-MEM [30]	9929.7	7350.8	5480.2	2967.9	2197.1	1638.0	10.071	13.604	18.247	33.694	45.515	61.050
GEM [34]	17771.4	15956.4	11872.8	5077.5	4559.0	3392.2	5.627	6.267	8.423	19.695	21.935	29.479
Hobbes3 [35]	5602.9	4432.4	2448.9	1742.4	1378.4	761.6	17.848	22.561	40.834	57.393	72.548	131.308
mrFAST [32]	87747.3	48188.4	46491.9	26170.2	14372.0	13866.0	1.140	2.075	2.151	3.821	6.958	7.212
RazerS [28]	17277.2	10413.1	7143.8	4841.2	2917.8	2001.7	5.788	9.603	13.998	20.656	34.272	49.957
RazerS3-P [29]	10462.5	6759.3	4900.3	3105.3	2006.2	1454.4	9.558	14.794	20.407	32.203	49.845	68.755
RazerS3-S [29]	13323.3	8192.2	5926.3	3830.9	2355.5	1704.0	7.506	12.207	16.874	26.104	42.453	58.686
Yara [33]	40988.1	25905.1	15463.1	11710.9	7401.4	4418.0	2.440	3.860	6.467	8.539	13.511	22.635
Proposed	8001.2	4803.2	2197.2	1488.5	893.6	408.8	12.498	20.820	45.512	67.181	111.912	244.644

**Normal and dynamic energy consumption (application energy) are indicated by Energy and Energy-Dy. Their values and the corresponding energy efficiencies with respect to reads mapped per joule energy consumption are reported in the table for different chromosomes. In RazerS3, -P and -S stands for pigeonhole and swift filters. Chr– chromosome.*

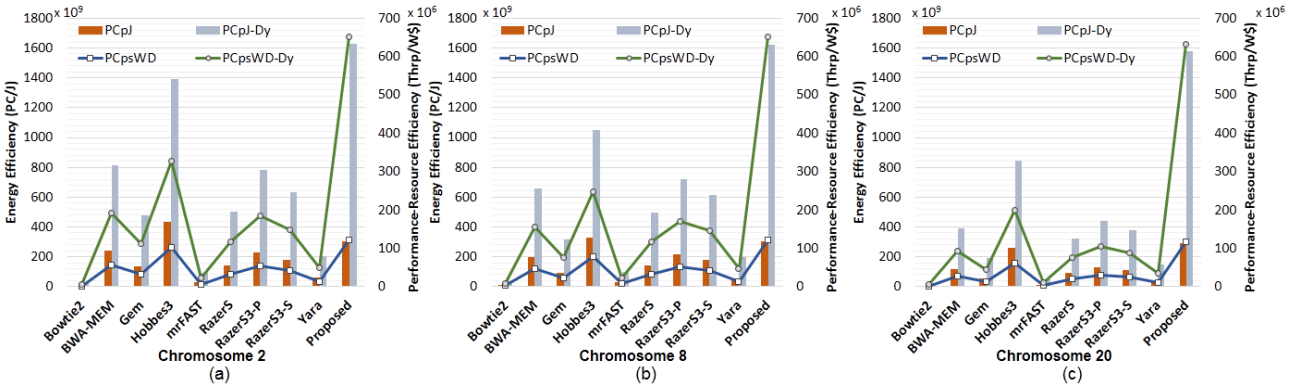

Fig. 9. Comparison of the proposed methodology for read mapping with software methods. The bar and line plots correspond to energy and performance-resource efficiencies, respectively. In RazerS3, -P and -S stands for pigeonhole and swift filters. Dy– is used to indicate measurements observed during dynamic power consumption. PCpJ: pairs compared per unit energy; PCpsWD: throughput per unit watt per $.

3.4–69.5× and 3.4–118.4× in dynamic conditions. On average, the gain over performance-resource efficiency is observed to be 13.3× and 20.65× in normal and dynamic conditions, respectively.

4.3.2 Efficiency in comparison with hardware methods

In this sub-section, the proposed methodology is evaluated with energy and resource efficiencies in comparison with state-of-the-art hardware methods. Due to the dependence on FPGA technology and architecture, it is difficult to make a direct comparison between two different FPGA designs. As RTL designs and files of most of the hardware methods are unavailable, it is impractical to implement them on our FPGA device. For example, the Gatekeeper [24] proposed by Alser et al. couldn't fit on the Zynq Ultrascale+ FPGA device chosen in this study as it occupied more than 134% LUTs. With respect to the methods in the literature, the primary concerns are energy and resource efficiencies. Consequently, we focus mainly on the energy and resource consumption of the read mapping application as a whole irrespective of the platform and hardware. Accordingly, all the hardware methods shall be assumed to be running on their respective platforms, as described in the literature.

The energy and resource performance metrics mentioned in the earlier sections shall be used in this comparative study. Few hardware methods require a computer system to run the whole operation of read mapping. A $1,000 computer system is considered for the sake of generality. The comparison is tabulated in Table 6, along with the information of experiments as reported in the literature. The power values are either obtained from the literature or from the reports after synthesizing the designs targeting their respective platforms. In some cases where neither power values nor design files are present, the FPGA resource utilization reported in the literature is used to estimate the power values using Xilinx Power Estimator. In some cases, if energy values are available, then these values are used. In the case of the hybrid systems, the power consumption of the CPU is estimated by the information provided in the respective articles using power calculators (http://powersupplycalculator.net, https://www.msi.com/calculator). The average power, *Pavg*, is calculated from the view of a simple calculation of energy consumption considering the respective execution time of hardware and CPU. The product of *Pavg* and time gives the total energy consumption. As an example, power and time values of [24] are estimated as follows.

By using the mapping performances of 16 core and single systems, the total time is estimated as 43.68 hours for the single core system. As the hardware accelerator consumes 5% of the total time, it takes 7861.893 seconds while the CPU takes 149375.971 seconds for mapping reads mentioned in Table 6. For average CPU power of 168 watt and hardware power of 12.5 watt, the *Pavg*, is (12.5×7861.893 + 168×149375.971)/ 157237.864 = 160.225 watt.

It is imperative to note that the PCpsWD metric highly depends on the cost of the hardware platform considered and this metric only gives a general idea of the efficiency. In the proposed design, reads are searched individually, one at a time. For a fair comparison, we have considered

TABLE 6
Energy Efficiency and Resource Performance of the Proposed Methodology in Comparison with Hardware Methods

Method	Platform,$Cost	Pavg (W)	Time (s)	Energy (J)	# Reads	Read Size	Reference Size	Energy Efficiency (PC/J)	Resource Performance (Thrp/W$)
Alser [24]	CPU + XC7VX690T,4995	160.225	157237.864	25.193 (M)	4093747	100	3,101,809,796	50.402	7.463
Benkrid [12]	Virtex-4 LX160,10000	139	1.871	260.069	1	256	141,218,456	0.139	0.014
Kierzynka [18]	Zynq XC7Z045,847	8.7	716572080	6234.18 (M)	1503238553	100	3,101,809,796	74.792	88.302
Cruz [41]	Zynq XC7Z100,1313	0.56	1.013	0.9868	NA	NA	453,920,704	0.46	0.609
Neves [42]	Zynq XC7Z020,895	0.289	0.068	0.0196	1	2276	4,092	0.475	0.560
Oliver [19]	CPU + XC6VLX240T,1995	110.66	224	24788.624	100000	50	222,389,117	44.857	14.977
Xin [43]	Virtex-6 XC6VLX365T,2495	6.377	37	235.949	1000000	36	1,000,000	152.575	61.152
Chen [44]	CPU+Virtex-5 XC5VLX330,1895	74.785	38	2841.859	1000000	100	4,938,920	173.792	91.711
Proposed	Zynq XCZU9EG, 2495	4.577	480	2197.2123	100000	100	64,444,167	293.299	117.555

**PC/J: pairs compared per joule, Thrp: throughput (PC/s). Platform indicates the hardware resources required to perform the entire operation of read mapping. All the FPGA devices are from Xilinx. Wherever required, a $1,000 computer system is considered for the sake of generality. Values of energy efficiency is in billion (10^9) and resource performance in million (10^6).*

TABLE 7
Comparison of Sequence alignment using Proposed Methodology and Hardware Methods

Method	Pavg (W)	Time (s)	Energy (J)	Query Size	Reference Size	CUPS (#/s)	CUPJ (#/J)	Resource Performance (Thrp/W$)
Benkrid [12]	39	1.871	72.969	256	141,218,456	19.322 (0.98)	495.442 (8.38)	0.049 (33.62)
Cruz [41]	0.56	1.013	0.9868	NA	453,920,704	0.448 (42.45)	460 (9.034)	0.609 (2.73)
Neves [42]	0.289	0.068	0.0196	2276	4,092	0.137 (138.88)	474.737 (8.75)	0.560 (2.97)
Proposed	4.577	0.011 m	0.05034 m	100	2,164	19.021 (1)	4155.888 (1)	1.665 (1)

**CUPS: number of cell updates per second (×10⁹). CUPJ: number of cell updates per joule (×10⁶). Thrp: throughput (PC/s, ×10⁶). The proposed method is used for alignment without considering the effect of filtering. The power value of the proposed method is for the entire system including filtering and verification. System designed only for alignment will consume lesser power. Time and energy values for the proposed method are in millisecond and milli-joule, respectively. Gain of proposed with respect to the method is given inside brackets.*

single-core values reported in [24]. It is seen that the proposed methodology is 1.68–2109.93× and 1.28-8396.78× more energy and resource efficient, respectively. As the methods presented in [12], [41], [42] are sequence alignments, a fair comparison is not maintained while PCpJ and PCpsWD metrics are used on the test data given in Table 6. In the next sub-section, these methods are compared with the proposed methodology using the metrics given in their respective articles. On excluding these methods, the gain over energy and resources obtained by the proposed methodology are 1.68–6.53× and 1.28–15.75×, respectively. On average, the gain over energy consumption and resource utilization is 3.97× and 5.62×.

4.3.3 Efficiency in comparison with hardware methods for sequence alignment

In the comparison of our methodology with the hardware methods proposed by Benkrid, Cruz and Neves [12], [41], [42], many of the reads are filtered out by the filtering stage in our design. The earlier defined metrics and performance results in Table 6 take this into account. In our design, the verification stage performs the alignment operation using the Myers bit-vector algorithm. Here, we compare the above alignment methods with the proposed verification. The proposed design is considered without the effect of the filtering stage. This design is then used to perform alignment of a 100 base-pairs size query sequence to a 2164 base-pairs size reference sequence. The results are tabulated in Table 7.

In this comparison, the dynamic power value for alignment, as reported in [12], is used. The performance and energy metrics defined as the number of cell updates per second (CUPS) and the number of cell updates per joule (CUPJ), respectively, are used in the comparison. CUPS and CUPJ are similar to the throughput (PC/s) and energy efficiency (PCpJ) metrics as defined earlier in section 4.3.1. In addition, the performance-resource metric PCpsWD is also studied for the new test data. The CUPS improvement of the proposed methodology in comparison with Benkrid *et al*. is less than unity. However, the gain over energy and resources is better. In comparison with all the methods, it is observed that the proposed methodology is more energy and resource efficient, to the tune of 8.38–9.03× and 2.73–33.62×, respectively.

In brief, the proposed methodology is the most energy-efficient in comparison with state-of-the-art software and hardware methods. Also, it produces more mappings per resources than other methods available in the literature. The features of better energy-efficiency and resource-performance of the proposed methodology can make low-cost genomics possible.

5 Conclusion and Future Work

In this paper, we propose a novel, low-cost solution for read mapping. A sorted q-gram lemma based parallel filtering methodology followed by an *in-situ* verification designed using Myers bit-vector algorithm are key to the proposed solution. The proposed methodology is implemented using a single Zynq Ultrascale+ XCZU9EG-2FFVB1156 FPGA device and thoroughly evaluated using real and simulated reads. The experimental results demonstrate clear gains in terms of energy and resource footprints while maintaining accuracy and full sensitivity when validated against the existing methods. Compared to state-of-the-art software and hardware methodologies, the proposed methodology achieves up to 7.8× more energy efficiency and up to 13.3× less resource utilization. We believe that the proposed methodology opens up new possibilities for low-cost and embedded genomic systems that may potentially find applications in healthcare.

As future work, we intend to optimize our architectures and improve the time performance of the proposed read mapping. We plan to design filtering and verification to work on multiple reads simultaneously. Additionally, in architectures with large section sizes, verification dominates the execution time of read mapping. The SQA can be investigated to include locations that can improve verification time by providing a small range of the reference genome rather than working on the entire section.

Acknowledgment

V. Y. Gudur's research is supported by the Visvesvaraya PhD Scheme for Electronics & IT by MeitY, Government of India (GOI) and partly by the IOT Based Holistic Prevention and Prediction of CVD (i-PREACT) project under ICPS Programme, Department of Science & Technology, GOI. S. Maheshwari's research is supported by EPSRC DTP scholarship at Newcastle University, UK. The authors would like to acknowledge the funding supports from the Royal Society Exchange (IE161183) and EPSRC IAA 'Whipserable' projects.

References

[1] E. Ayday, E. De Cristofaro, J. Hubaux and G. Tsudik, "Whole Genome Sequencing: Revolutionary Medicine or Privacy Nightmare?," in *Computer*, vol. 48, no. 2, pp. 58-66, Feb. 2015.
[2] C. Auffray, D. Charron, and L. Hood, "Predictive, preventive,

personalized and participatory medicine: Back to the future," *Genome Med.*, vol. 2, no. 8, pp. 8–10, 2010.
[3] C. S. Bloss, B. F. Darst, E. J. Topol, and N. J. Schork, "Direct-to-consumer personalized genomic testing," *Hum. Mol. Genet.*, vol. 20, no. R2, pp. 132–141, 2011.
[4] A. Al Kawam, "Understanding the Bioinformatics Challenges of Integrating Genomics Into Healthcare," *IEEE J. Biomed. Heal. Informatics*, vol. 22, no. 5, pp. 1672–1683, 2018.
[5] Z. D. Stephens *et al.*, "Big data: Astronomical or genomical?," *PLoS Biol.*, vol. 13, no. 7, pp. 1–11, 2015.
[6] G. P. Consortium, "A map of human genome variation from population-scale sequencing," *Nature*, vol. 467, pp. 1061-1073, 2010.
[7] B. Schmidt and A. Hildebrandt, "Next-generation sequencing: big data meets high performance computing," *Drug Discov. Today*, vol. 22, no. 4, pp. 712–717, 2017.
[8] J. A. Reuter, D. V. Spacek, and M. P. Snyder, "High-Throughput Sequencing Technologies," *Mol. Cell*, vol. 58, no. 4, pp. 586–597, 2015.
[9] H. Ye, J. Meehan, W. Tong, and H. Hong, "Alignment of short reads: A crucial step for application of next-generation sequencing data in precision medicine," *Pharmaceutics*, vol. 7, no. 4, pp. 523–541, 2015.
[10] A. Ghosh, "The big push for renewable energy in India: What will drive it?," *Bull. At. Sci.*, vol. 71, no. 4, pp. 31–42, 2015.
[11] C. B. Olson *et al.*, "Hardware acceleration of short read mapping," *Proc. 2012 IEEE 20th Int. Symp. Field-Programmable Cust. Comput. Mach. FCCM 2012*, pp. 161–168, 2012.
[12] K. Benkrid, A. Akoglu, C. Ling, Y. Song, Y. Liu, and X. Tian, "High performance biological pairwise sequence alignment: FPGA versus GPU versus cell BE versus GPP," *Int. J. Reconfigurable Comput.*, vol. 2012, 2012.
[13] S. Aluru and N. Jammula, "A review of hardware acceleration for computational genomics," *IEEE Des. Test*, vol. 31, no. 1, pp. 19–30, 2014.
[14] E. J. Houtgast, V. M. Sima, G. Marchiori, K. Bertels, and Z. Al-Ars, "Power-efficiency analysis of accelerated BWA-MEM implementations on heterogeneous computing platforms," *2016 Int. Conf. Reconfigurable Comput. FPGAs, ReConFig 2016*, pp. 1–8, 2016.
[15] J. Arram, T. Kaplan, W. Luk, and P. Jiang, "Leveraging FPGAS for accelerating short read alignment," *IEEE/ACM Trans. Comput. Biol. Bioinforma.*, vol. 14, no. 3, pp. 668–677, 2017.
[16] N. Cadenelli, Z. Jakšić, J. Polo, and D. Carrera, "Considerations in using OpenCL on GPUs and FPGAs for throughput-oriented genomics workloads," *Futur. Gener. Comput. Syst.*, vol. 94, pp. 148–159, 2019.
[17] S. S. Banerjee *et al.*, "ASAP: Accelerated Short-Read Alignment on Programmable Hardware," *IEEE Trans. Comput.*, vol. 68, no. 3, pp. 331–346, 2019.
[18] M. Kierzynka, L. Kosmann, and S. Krupop, "Energy efficiency of sequence alignment tools — Software and hardware perspectives," *Futur. Gener. Comput. Syst.*, 2016.
[19] O. Knodel, T. B. Preusser, and R. G. Spallek, "Next-generation massively parallel short-read mapping on FPGAs," *Proc. Int. Conf. Appl. Syst. Archit. Process.*, pp. 195–201, 2011.
[20] V. Y. Gudur and A. Acharyya, "Hardware-Software Codesign based Accelerated and Reconfigurable Methodology for String Matching in Computational Bioinformatics Applications," *IEEE/ACM Trans. Comput. Biol. Bioinforma.*, pp. 1–14, 2018.
[21] H. Xin *et al.*, "Shifted Hamming distance: A fast and accurate SIMD-friendly filter to accelerate alignment verification in read mapping," *Bioinformatics*, vol. 31, no. 10, pp. 1553–1560, 2015.
[22] H. Li and N. Homer, "A survey of sequence alignment algorithms for next-generation sequencing," *Brief. Bioinfo.*, vol. 11,no. 5,pp. 473–483, 2010.
[23] Y. Chan, K. Xu, H. Lan, W. Liu, Y. Liu and B. Schmidt, "PUNAS: A Parallel Ungapped-Alignment-Featured Seed Verification Algorithm for Next-Generation Sequencing Read Alignment," *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Orlando, FL, 2017, pp. 52-61.
[24] M. Alser, H. Hassan, H. Xin, O. Ergin, O. Mutlu, and C. Alkan, "GateKeeper: a new hardware architecture for accelerating pre-alignment in DNA short read mapping," *Bioinformatics*, vol. 33, no. 21, pp. 3355–3363, 2017.
[25] S. Burkhardt A. Crauser P. Ferragina H.-P. Lenhof E. Rivals M. Vingron, "Q-gram based database searching using a suffix array (QUASAR)," *Proc. Int. Conf. Computat. Molec. Biol.*, pp. 77-83, 1999.
[26] K. Rasmussen J. Stoye E. W. Myers, "Efficient q-Gram Filters for Finding All epsilon-Matches Over a Given Length," *Journal of Computational Biology*, vol. 13, pp. 296-308, 2006.
[27] G. Myers, "A Fast Bit-Vector Algorithm for Approximate String Matching Based on Dynamic Programming," *Journal of the ACM*, vol. 46, no. 3, pp. 395–415, 1999.
[28] D. Weese, A. Emde, T. Rausch, and A. Do, "RazerS — fast read mapping with sensitivity control," *Genome Research*, vol. 19, no. 9, pp. 1646–1654, 2009.
[29] D. Weese, M. Holtgrewe, and K. Reinert, "Sequence analysis RazerS 3 : Faster , fully sensitive read mapping," *Bioinformatics*, vol. 28, no. 20, pp. 2592–2599, 2012.
[30] H. Li and R. Durbin, "Fast and accurate long-read alignment with Burrows – Wheeler transform," *Bioinforma*, vol. 26, no. 5,pp. 589–595, 2010.
[31] B. Langmead and S. L. Salzberg, "Fast gapped-read alignment with Bowtie 2," *Nat. Methods*, vol. 9, no. 4, pp. 357–359, 2012.
[32] C. Alkan *et al.*, "Personalized copy number and segmental duplication maps using next-generation sequencing," *Nat. Genet.*, vol. 41, no. 10, pp. 1061–1067, 2009.
[33] E. Siragusa, "Approximate string matching for high-throughput sequencing," Free University of Berlin, 2015.
[34] S. Marco-Sola, M. Sammeth, R. Guigó, and P. Ribeca, "The GEM mapper: Fast, accurate and versatile alignment by filtration," *Nat. Methods*, vol. 9, no. 12, pp. 1185–1188, 2012.
[35] J. Kim, "Hobbes3 : Dynamic Generation of Variable-Length Signatures for Efficient Approximate Subsequence Mappings," *2016 IEEE 32nd Int. Conf. Data Eng.*, pp. 169–180, 2016.
[36] M. David, M. Dzamba, D. Lister, L. Ilie, and M. Brudno, "SHRiMP2: Sensitive yet practical short read mapping," *Bioinformatics*, vol. 27, no. 7, pp. 1011–1012, 2011.
[37] S. Maheshwari, V. Y. Gudur, R. Shafik, I. Wilson, A. Yakovlev, and A. Acharyya, "CORAL: Verification-aware OpenCL based Read Mapper for Heterogeneous Systems," *IEEE/ACM Trans. Comput. Biol. Bioinforma.*, vol. 14, no. 8, pp. 1–1, 2019.
[38] K. Reinert, B. Langmead, D. Weese, and D. J. Evers, "Alignment of Next-Generation Sequencing Reads," *Annu. Rev. Genomics Hum. Genet.*, vol. 16, no. 1, pp. 133–151, 2015.
[39] R. Baeza-Yates and G. Navarro, "Faster Approximate String Matching," *Algorithmica*, vol. 23, no. 2, pp. 127-158, 1999.
[40] H. Cheng, H. Jiang, J. Yang, Y. Xu, and Y. Shang, "BitMapper: An efficient all-mapper based on bit-vector computing," *BMC Bioinformatics*, vol. 16, no. 1, 2015.
[41] M. T. Cruz, P. Tomás, and N. Roma, "Energy-efficient architecture for DP local sequence alignment: Exploiting ILP and DLP," *Lect. Notes Comput. Sci.*, vol. 9044, pp. 194–206, 2015.
[42] N. Neves *et al.*, "Multicore SIMD ASIP for Next-Generation Sequencing and Alignment Biochip Platforms," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 23, no. 7, pp. 1287–1300, 2015.
[43] Y. Xin *et al.*, "Parallel architecture for DNA sequence inexact matching with Burrows-Wheeler Transform," *Microelectronics J.*, vol. 44, no. 8, pp. 670–682, 2013.
[44] Y. Chen, B. Schmidt and D. L. Maskell, "A hybrid short read mapping accelerator," *BMC Bioinform.*, vol. 14, no. 1, pp. 67, 2013.
[45] H. C. Ng, S. Liu, and W. Luk, "Reconfigurable acceleration of genetic sequence alignment: A survey of two decades of efforts," *2017 27th Int. Conf. F. Program. Log. Appl. FPL 2017*, 2017.
[46] C. Wang, X. Li, P. Chen, A. Wang, X. Zhou, and H. Yu, "Heterogeneous cloud framework for big data genome sequencing," *IEEE/ACM Trans. Comput. Biol. Bioinforma.*, vol. 12, no. 1, pp. 166–178, 2015.
[47] E. B. Fernandez, J. Villarreal, S. Lonardi and W. A. Najjar, "FHAST: FPGA-Based Acceleration of Bowtie in Hardware," in *IEEE/ACM Transactions on Comput. Bio. and Bioinform.*, vol. 12, no. 5, pp. 973-981, 1 Sept.-Oct. 2015.
[48] P. Chen, C. Wang, X. Li, and X. Zhou, "Accelerating the next generation long read mapping with the FPGA-based system," *IEEE/ACM Trans. Comput. Biol. Bioinforma.*, vol. 11, no. 5, pp. 840–852, 2014.
[49] D. Z. Chen, "Efficient Parallel Binary Search on Sorted Arrays, with Applications," *IEEE Trans. Parallel Distrib. Syst.*, vol. 6, no. 4, pp. 440–445, 1995.
[50] M. Santarini, "Zynq-7000 EPP Sets Stage for New Era of Innovations", *Xcell journal*, no. 75, pp. 8-13, 2011.
[51] J. Teich, "Hardware/Software Codesign: The Past, the Present, and Predicting the Future", *Proceedings of the IEEE*, vol. 100, no., pp. 1411-1430, 2012.
[52] M. Holtgrewe, "Mason – A Read Simulator for Second Generation Sequencing Data," Tech. Rep. TR-B-10-06, Free University of Berlin, 2010.
[53] J. S. Kim *et al.*, "GRIM-Filter: Fast seed location filtering in DNA read mapping using processing-in-memory technologies," *BMC Genomics*, vol. 19, no. Suppl 2, 2018.
[54] M. Alser *et al.*, "Accelerating genome analysis: A primer on an ongoing journey," *arXiv*, no. October 2020, pp. 65–75, 2020.
[55] D. S. Cali *et al.*, "GenASM: A high-performance, low-power approximate string matching acceleration framework for genome sequence analysis," *Proc. Annu. Int. Symp. Microarch., MICRO*, vol.2020-Oct.,pp. 951–966, 2020.
[56] Y. Turakhia, G. Bejerano, and W. J. Dally, "Darwin: A genomics co-processor provides up to 15,000× acceleration on long read assembly," *ACM SIGPLAN Not.*, vol. 53, no. 2, pp. 199–213, 2018.
[57] H. Hyyrö, "A bit-vector algorithm for computing Levenshtein and Damerau edit distances," *Nord. J. Comput.*, pp. 1–11, 2003.
[58] M. Holtgrewe, A. K. Emde, D. Weese, and K. Reinert, "A novel and well-defined benchmarking method for second generation read mapping," *BMC Bioinformatics*, vol. 12, 2011.

Venkateshwarlu Y. Gudur is working towards the Ph.D. degree in Microelectronics and VLSI at the Department of Electrical Engineering, Indian Institute of Technology (IIT) Hyderabad, India. His research interests include hardware acceleration in healthcare applications, VLSI, SoC and reconfigurable computing.

Sidharth Maheshwari is a final year Ph.D. student at the School of Engineering, Newcastle University, UK. He is working on energy-efficient and performance-driven embedded genomics solutions to computational pipelines of whole genome sequencing. His research interests include Bioinformatics, VLSI and Biomedical Engineering.

Amit Acharyya received the Ph.D. degree from the School of Electronics and Computer Science, University of Southampton, U.K., in 2011. He is currently an Associate Professor with IIT Hyderabad, India. His research interests include signal processing algorithms, VLSI architectures, low power design techniques, bioinformatics.

Rishad Shafik (MIET, SMIEEE) is an Associate Professor (Senior Lecturer) of Electronic Systems within the School of Engineering, Newcastle University, UK. He received his Ph.D. from Southampton in 2010. He is the author/co-author of 130+ IEEE/ACM journal and conference articles, with three best paper nominations. His research interests include energy-efficiency and autonomy aspects of embedded computing systems.