Journal of Signal Processing Systems (2020) 92:1197–1213
https://doi.org/10.1007/s11265-019-01452-x

Towards Accelerated Genome Informatics on Parallel HPC Platforms: The ReneGENE-GI Perspective

Santhi Natarajan[1] · Krishna Kumar N.[1] · Debnath Pal[1] · S. K. Nandy[1]

Received: 25 September 2018 / Revised: 5 January 2019 / Accepted: 2 April 2019 / Published online: 23 April 2019

Abstract
Genome Informatics (GI) involves accurate computational investigations of strongly correlated subsystems that demands inter-disciplinary approaches for problem solving. With the growing volume of genomic sequencing data at an alarming rate, High Performance Computing (HPC) solutions offer the right platform to address the computational needs. GI requires algorithm-architecture co-design of parallel and accelerated biocomputing involving reconfigurable hardware like FPGAs and graphics accelerators or GPUs, to bridge the gap between growing data volumes and compute capabilities. Such platforms offer high degrees of parallelism and scalability, while accelerating the multi-stage GI computational pipeline. Amidst such high computing power, it is the choice of algorithms and implementations in the entirety of the GI pipeline that decides the precision of bio-computing in revealing biologically relevant information. Through this paper, we present ReneGENE-GI, an innovatively engineered GI pipeline. This paper details the performance analysis of ReneGENE-GI's Comparative Genomics Module (CGM), the compute intensive stage of the pipeline. This module comes in two flavours, designed to run on GPUs and FPGAs respectively, hosted on HPC platforms. The pipeline uses a very efficient reference indexing algorithm based on the dynamic Monotonic Minimal Perfect Hashing Function (MMPH), allowing an absolute indexing for the reference genome, thus avoiding heuristics. Alignment time for our FPGA version is about one-tenth the time taken by our single GPU implementation, which itself is 2.62x faster than CUSHAW2-GPU (the GPU CUDA implementation of CUSHAW). With the single-GPU implementation demonstrating a speed up of 150+ x over standard heuristic aligners in the market like BFAST, the FPGA version of our CGM is several orders faster than the competitors, offering precision over heuristics.

Keywords Genome informatics · High performance computing · Reconfigurable hardware · GPU · FPGA · Accelerator hardware · NGS · Short read mapping · Sequencing

1 Introduction

Embedded in a long string spanning several billion characters, drawn from a set of genetic alphabets, the genomic big data encompasses a well authored genetic literary work that narrates the story of evolution over billions of years. Genome Informatics (GI), the study of genomes, integrates the big data of genomes with a ubiquitous base of interoperable medical and engineering disciplines. GI has evolved to be a discovery-driven approach to analyse the unstructured genomic big data, which takes inferences from an organism's genetic code to arrive at translationally important interpretations. Upcoming and widely popular GI applications cater to numerous domains including targeted personalized diagnostics and therapeutics, thereby improving the effectiveness of healthcare.

1.1 Sequencing for GI

Understanding the genome through GI involves determining the order of the genetic alphabets or bases, namely adenine

✉ Santhi Natarajan
santhin@iisc.ac.in

Krishna Kumar N.
kkrishna@iisc.ac.in

Debnath Pal
dpal@iisc.ac.in

S.K. Nandy
nandy@iisc.ac.in

[1] Indian Institute of Science, Bangalore, 560012, India

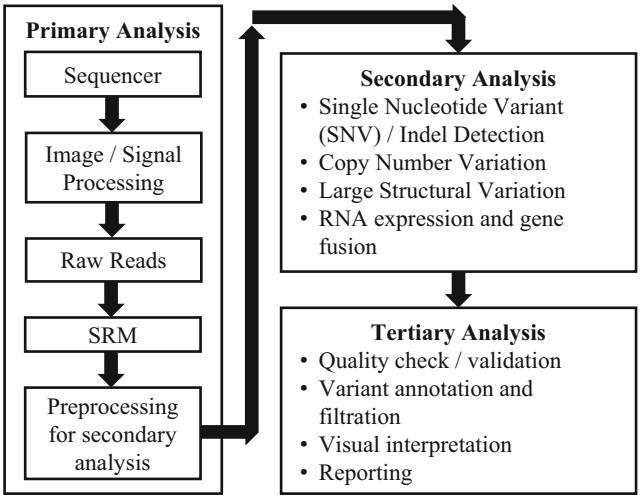

Figure 1 NGS workflow.

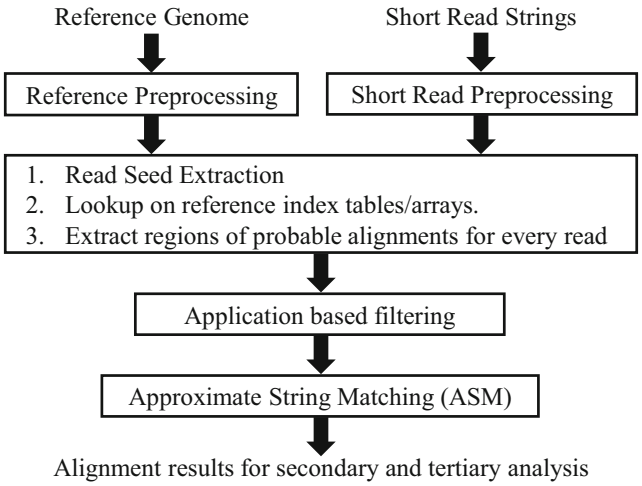

Figure 2 SRM workflow.

(A), cytosine (C), guanine (G) and thymine (T), within the genomic sequence, and the process is widely known as sequencing. Next Generation Sequencing (NGS) involves massively parallel sequencing of genetic data with high throughput, while offering an unparalleled interrogation of the genome, throwing deeper insight into the functional and structural investigation of genetic data [1, 2].

Data processing with NGS, over an elaborated multi-stage data-analytics pipeline, is depicted in Fig. 1. At the end of the primary data analysis, the pipeline generates several intermediate files and output files of significant magnitude, contributing to petabytes of NGS big data of raw sequence short reads per sample per run. Each short read is a very small fragment or substring of the target genome string under consideration. The short reads are then aligned or mapped to a reference genome string through a process called Short Read Mapping (SRM) or Short Read Alignment (SRA). By the year 2025, genomic data acquisition through NGS, being highly geographically distributed across multiple species, is predicted to reach the rate of one zettabase per year [3].

1.2 Short Read Mapping (SRM): Computational Bottleneck in GI

The SRM process, illustrated in Fig. 2, is interpreted as a classic Approximate String Matching (ASM) problem. SRM attempts to search the specific short read string q of length —q— (ranging from about 25 to a few hundred bases), over a much longer reference genome string G of length —G— (a human reference genome is typically 3 billion bases long). SRM aims to find the regions of origin of each short read string with respect to the reference, and hence finds regions of similarity or dissimilarity, over the character set $\Sigma = \{A, C, G, T\}$ [4, 5].

SRM performs ASM for its input strings, with a cost function and appropriate error model to accommodate errors in strings induced due to:

1. Genetic mutations (computationally interpreted as a character being replaced with another over the character set Σ)
2. Single Nucleotide Polymorphism (SNP) (algorithmically similar to genetic mutations, but interpretations vary biologically over a sample population)
3. Insertions or deletions of bases (computationally interpreted as addition or deletion of a genetic character across the length of the string)
4. Other evolutionary genetic alterations
5. NGS platform induced errors

The algorithms here quantify the similarity or the edit distances between the two strings under consideration. Edit distance is the minimum number of more likely deviations or manoeuvres that can transmute one string to another. The cost function in ASM assigns each such manoeuvre a cost, and eventually aims at minimizing or maximizing the total cost based on the limits of the cost function, which serves to quantify similarity between the two strings.

2 Related Work: SRM in GI Using Accelerators and HPC Platforms

With growing volume of NGS big data, the SRM and subsequent analytic steps demand an HPC environment complimented with accelerators for data storage and analyses [6, 7]. NGS has thus become a complex engineering problem, eliciting innovative computational, scientific and statistical approaches towards big data analysis. A strict validation of various algorithms and

Table 1 Alignment methods.

Steps	Method 1	Method 2
1	Compute alignment matrix D with FPGA/GPU	Compute matrix D and optimal score with FPGA/GPU
2	Transfer D to host	Transfer optimal score to host
3	Perform traceback in the host	Filter sequence pairs with scores exceeding threshold
4	Extract final optimal alignment	Recompute matrix D in the host for filtered pairs
5	–	Perform traceback in the host and extract final optimal alignment for filtered pairs
Drawback	Severe memory bottleneck due to large matrix data for longer sequences	Matrix fill stage takes 98.6% of the total alignment time, more compute intensive and complex, with repetition of calculations to achieve alignment.

softwares in an NGS pipeline is essential, to ensure reliable and accurate results [8–11].

2.1 Accelerators for SRM

The most popular and central scheme for SRM is the Dynamic Programming (DP) methodology. Though computationally complex, the DP algorithms prove to be very efficient in discriminating substantial similarities amongst severe noise in genetic data presented by evolution. There are several parallel implementations of DP method [12–17]. While some adopt parallel computations using SIMD (Single Instruction Multiple Data) style instructions within a single processor, others realize parallelism on multiple processors. There are various accelerator platforms like reconfigurable hardware (FPGAs) and GPUs on which the DP recursive equation kernel is realized as multiple threads or blocks to accelerate alignment. Most of these methods can be classified into two major categories listed in Table 1. We can see the bottlenecks offered by these methods, thus rendered not useful while handling big data.

2.2 SRM on HPC Platforms

To perform SRM on such large data volumes, GI adopts a multi-stage multi-algorithmic parallel pipeline. The deployment of the GI pipeline exploits the best practices in HPC on platforms like clouds, grids, accelerators and clusters, while strictly following bio computational principles in classical genetics, molecular and cell biology. All such efforts are predominantly directed towards prospecting the unexploited scope of parallelism and scalability of the HPC platforms [18, 19].

However, the bio-computing within the GI pipeline is irregular and combinatorial in nature. It is irregular due to being heavily data dependent, lacking sense of temporal and spatial locality of data. This severely curbs the performance of modern processor architectures built on deep memory hierarchies meant for pertinent data structures. The runtime computational irregularities are perfectly complemented by the non contiguous file accesses, making an optimal parallelization of GI pipeline on a multi-core environment more difficult. The big data along with an all-to-all computation contributes to the time and computational complexity of the combinatorial algorithms. This makes fine-grain synchronization an utmost necessity to exploit data-level and process-level parallelism in a multi-node and multi-core HPC environment. In presence of a variety of accelerator platforms to conceive the parallel versions of the various computational algorithms, a substantial engineering effort is required in optimizing bio-computing on the available HPC hardware for concurrency, time, cost, and coverage [6, 9, 20, 21].

Through this paper, we present the detailed performance analysis of ReneGENE-GI, an innovatively engineered GI pipeline. The architecture of ReneGENE-GI was discussed in our previous work [22]. This paper is an extension of the same, with more algorithmic, performance and implementation details of the various stages of the pipeline. It performs mapping of raw genomic data from the NGS platforms with high precision. The pipeline hosts a unique blend of highly dynamic multi-dimensional data structures and parallel algorithms designed for executing the irregular genomic computing on accelerator based hardware and HPC platforms. ReneGENE-GI exploits the inherent parallelism and scalability of the hardware at the level of micro and system architecture to offer a reliable mapping for any NGS read data, regardless of the size. This allows for optimizing time, cost, and affordability without unduly penalizing biological fidelity of the results. It exploits a substantial degree of latent parallelism by engaging fine-grain synchronization, while allowing the application to scale up on HPC platforms.

The principal novelty of our solution involves engineering of the pipeline using existing algorithms on platforms using a data streaming approach that minimizes heap memory footprint and input/output bottlenecks. It is also supplemented by compiler-level and architecture-specific optimizations to improve the performance in a reconfigurable HPC environment. We also present the performance

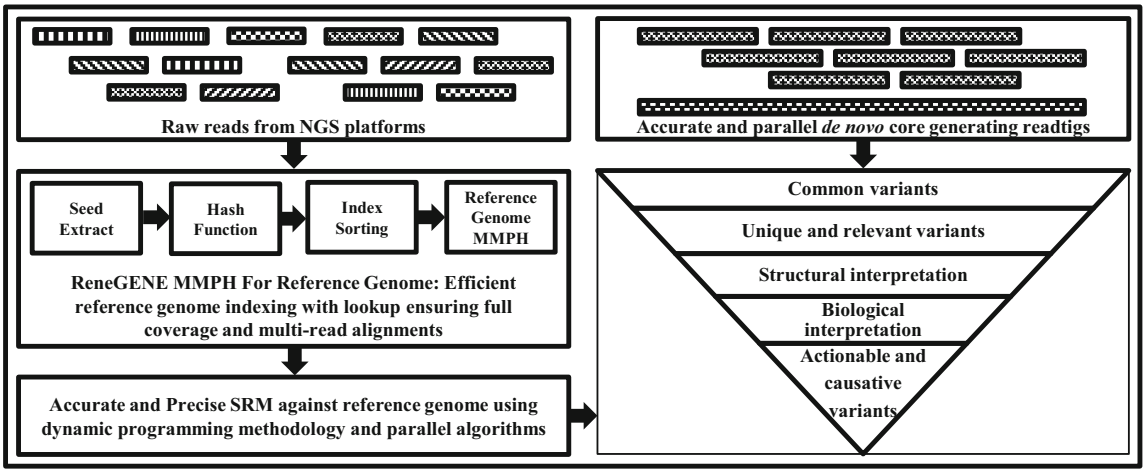

Figure 3 The ReneGENE-GI pipeline.

analysis for ReneGENE-GI's Comparative Genomics Module (CGM), implemented on both FPGA and GPU accelerator platforms.

3 The ReneGENE-GI Pipeline

The ReneGENE-GI pipeline, illustrated in Fig. 3, adopts the multi-stage NGS workflow illustrated in Fig. 1 for data analytics. While implementing the regular bio-computing algorithms used for SRM and subsequent steps, the pipeline follows a modular approach for each step and each algorithm in an effort to deploy the respective stages on an HPC environment to enable parallel computing.

3.1 Overview

The novelty of the ReneGENE-GI pipeline lies in the fact that it offers a unique blend of comparative genomics and *de novo* sequence assembly, offering the most precise SRM. The CGM exploits parallel dynamic programming methodology to accurately map the short reads against the reference genome. The alignment is backed by an exhaustive indexing and lookup of reads against the reference using the parallel implementation of dynamic Monotonic Minimal Perfect Hashing (MMPH) method [23]. This is a complete index of the reference, where the k-mer seeds fully cover the entire span of the reference, inclusive of the repeat regions. As compared to other indexing techniques that employ heuristics of purging repeat region hits, ReneGENE-GI pipeline reports those hits as well, throwing light on many anomalies embedded in these repeats.

The *de novo* module is implemented as a parallel map-reduce based readtig generation technique. The readtigs are extended short reads, based on a novel read extension algorithm, prototyped and verified for precision on HPC platforms with reconfigurable accelerator support. The readtigs are further mapped on to the reference genome to encompass the possible insertions and deletions of genetic alphabets at certain locations, thereby widening the map space and coverage.

The final SRM alignment results are then subjected to variant calling or preliminary tertiary analysis.

3.2 Reference Preprocessing in ReneGENE-GI

With an extremely long reference sequence string, indexing the reference over the alphabet set of $\Sigma = \{A, C, G, T\}$ is a difficult task. ReneGENE-GI presents an efficient indexing scheme for the reference. Here, we generate a hash table for the index, based on a static set of lexicographically ordered keys.

It is known that, a Perfect Hash Function (PHF), for a set U, places the keys from U in an index table for efficient lookup operations, by mapping distinct elements in U to distinct values, avoiding any collisions. The table is indexed by the output of the PHF. Such PHFs are best suitable for indexing, where the data is very large, and is less frequently updated. This method is space-efficient, where the table created is compact, for a static set of keys.

A PHF becomes a minimal PHF, when the PHF maps k keys to consecutive integer values, usually ranging from 1 to k or 0 to $k-1$. A minimal PHF is order preserving, when the keys are given in some order like $k_1, k_2, \ldots, k_n$, and for any two keys k_i and k_j, where $i < j \rightarrow PHF(k_i) < PHF(k_j)$.

A minimal PHF becomes Monotonic Minimal PHF (MMPH), when the lexicographical order of the keys is preserved. Now, considering an application like genomics, there is dynamics involved in the form of the continuous insertions and deletions into the set U. Hence, to avoid

heuristics in lookup, ReneGENE-GI implements a dynamic MMPH. This is typically done as part of preprocessing the reference genome, and coming up with index tables, over the lexicographically sorted set of keys extracted from the reference.

ReneGENE-GI indexing is illustrated in Algorithm 1. The keys are typically substrings of length k, called k-mer seeds, extracted by a sliding window operation on the genome. By performing a dynamic MMPH, Rene-GENE-GI maps each k-mer from a lexicographically ordered keyset K, to its corresponding index position in the Reference Index Table (RIT). The number of keys is always fixed, based on the choice of k. The natural order is always preserved over the keyset k by the binary encoding of bases in substring k (i.e., A $\rightarrow$ 00, C $\rightarrow$ 01, G $\rightarrow$ 10, T $\rightarrow$ 11). This algorithm renders a small memory footprint for the resulting reference index. For example, the human genome reference version GRCh38 (3.1 GB) was indexed in about 5 minutes using the MPI based implementation of Algorithm 1, generating an index of 5.4 GB in size.

Algorithm 1 RIT_MMPH().

// **Purpose:** MMPH Function for reference genome string to form RIT
// **Input:** Reference genome G, hashing function parameters
// **Output:** RIT Hash Table
- -
Load Reference Genome G
procedure FORM_SEED_KEYSET(G)
    sliding_window_function(G) ▷ to generate set U which is a natural lexicographically ordered set of k-mer keys, with left zero padded binary representation
end procedure
procedure BIN_PARTITION()
    Partition K into 2^b bins where $K \subseteq U$
    Bin $B_i \leftarrow \{x \in K \mid p(x) = i\}$, where $0 \leq i \leq 2^b - 1$ and $p(x)$ is the partition function, $p \in \{0, 1\}^b$ for some integer b
end procedure
for each B_i in bin set **do**
    for each key x in B_i **do**
        Bijective MMPH function h(x)
        $h : K \rightarrow s$, iff $x \leq y \Rightarrow h(x) \leq h(y), \forall x, y \in K and K \subseteq U$
        Update RIT hash table T_i
    end for
end for

The values corresponding to each position in the index, is the list of RIT IDs, which are locations corresponding to the occurrence of the k-mer seed across the length of the reference string. These values can be retrieved from the table by a single access to the table, thus searching the sorted index table with *O(1)* accesses to the table per key. The lookup process per read on the RIT table is explained in Algorithm 2.

In the context of repeat regions, a k-mer is extracted from multiple locations over a fragment of the reference, resulting in an extended list of values in the RIT. As compared to other indexing techniques that employ heuristics of purging repeat region hits, ReneGENE-GI reports those hits. These can eventually throw light on many anomalies embedded in the repeats, during alignment.

In an attempt to make the lookup process mutation aware, all possible mutations from all the locations of a single k-mer key is derived, and a lookup is performed for each of these mutation-induced k-mers. This results in a complete lookup, where instead of a single key, lookup is performed for a complete set of mutation-aware keys, for each read.

Algorithm 2 ReneGENE-GI_RIT_Lookup().

// **Purpose:** To perform lookup on RIT Hash Table for a short read and extract regions of probable alignment
// **Input:** Reference genome G, short reads, RIT Hash Tables
// **Output:** Regions of probable alignments per read
- -
Load Reference Genome G
Load short reads, $R \in q_1, q_2, ..., q_r$
for each read q_i in R **do**
    Extract k-mer seed set
    for each seed in seed set **do**
        Extract bin set
        for each bin B_i in bin set **do**
            Load RIT hash table T_i for B_i
            Lookup in T_i
            Extract RIT IDs for a hit in T_i
            for each RIT ID in RIT ID array **do**
                Extract region of probable alignment in G
                ASM_Qualify() ▷ Perform filtering for allowed error bound E
            end for
        end for
    end for
end for

3.3 SRM in ReneGENE-GI

The ReneGENE-GI pipeline performs SRM where the small fragments of genome from the NGS platforms, generally known as short reads, are mapped or aligned against a

reference genome string. SRM works on the massive input data set of short reads, typically of the order of petabytes, and aims to find the region of origin of each short read string with respect to the reference, and hence find regions of similarity or dissimilarity. Eventually, SRM builds the longer genome from the short reads, by putting short reads together as in a jig-saw puzzle, with respect to a reference genome. SRM is interpreted as a typical ASM problem in the ReneGENE-GI pipeline, to find occurrences of a smaller short read in a much larger reference [4, 24, 25].

Algorithm 3 ReneGENE-DP().

```
// Purpose: Approximate String Matching for pairwise sequence alignment with DP algorithms
// Input: short reads q, reference genome sub string g, DP algorithm parameters
// Output: Final optimal alignment score
-----------------------------------------
for Each Read in Read Set do
    Load Read q
    Load Reference genome substrings set, post lookup
    for Each g in reference substrings set do
        Initialize Alignment Matrix D
        Align q to gaps D[0][j] = 0          ▷ 0 ¡ j ¡ N
        Align g to gaps D[i][0] = 0          ▷ 0 ¡ i ¡ M
        Score_opt ← 0     ▷ Initialize optimal score to 0
        Intialize Insertion Matrix I;
        for i = 1 to M do
            for j = 1 to N do     ▷ recursive scoring model with affine gap penalty
                DP_SRM_Recursive_Function()
                Update Score_opt
            end for
        end for
        Score_opt ← max_{i=1,j=1}^{M,N} (D[i][j])
        Call_Traceback()
    end for
end for
```

The SRM, based on a Dynamic Programming (DP) [26] method, with preprocessing, is shown in Algorithm 3. While handling genome sequences, the DP technique is proven to be the most sensitive in performing ASM. The DP method comes with a quadratic time and space complexity of $O\ (LN)$. The DP based algorithms employ a recursive scoring or cost function model, with an appropriate linear or affine penalty model (for the dissimilarities and string errors), to assign scores for mapping. The algorithm adopts a matrix space, called the alignment matrix, D.

The SRM module of ReneGENE-GI runs on accelerator hardware like FPGAs and GPUs which are plugged in to the HPC systems. The ReneGENE-DP algorithm within the SRM module is designed to run as multiple parallel threads on the accelerator hardware. This effectively speeds up the entire pipeline providing multi-fold performance improvement over the state-of-the-art SRM implementations.

3.4 Read Extension Module of ReneGENE-GI

The *de novo* read extension module of the pipeline, deals with the problem of grouping the short reads based on an overlap relationship among the reads, in the absence of a reference genome. This algorithm is discussed in detail in our previous work [27]. Related reads are grouped together and they grow to form longer sequences called readtigs. Here again, a single read can share a similar overlap relationship with several of its sequence neighbours, resulting in a single seed growing into many readtigs. This is decided on run time and hence the computations are clearly irregular due to the irregularity in the relationships among the input data sets. To accommodate the readtigs or extended reads that grow on the fly, this module implements dynamically growing data structures cast in the map-reduce framework, allowing a parallel deployment. The *de novo* module processing is shown in Algorithm 4.

Algorithm 4 ReneGENE_Novo_Readtig().

```
// Purpose: de novo read extension function to generate readtigs
// Input: Input short reads R in fastq format
// Output: Assembled Readtigs C in fastq format
---------------------------------------------
Partition the short reads R into S_n read sets
for each read set S_i in S_n do
    Load each M_i in M_n readtig maps with reads from S_i to form readtig seeds
end for
for each readtig map M_i in M_n parallel maps do
    for each read r in R in the forward direction do
        if r overlaps with a readtig seed from M_i then
            Assemble r with the seed
        end if
    end for
    for each read r in R in reverse direction do
        if r overlaps with a readtig seed from M_i then
            Assemble r with the seed
        end if
    end for
end for
for each readtig map M_i in M_n do
    Merge contents to form readtig set C
end for
```

3.5 Variant Calling in ReneGENE-GI

Variants or mutations in a genome sequence represent the unique changes in genomic alphabets along the length of the target genome, with respect to a reference genome at specific locations. Variant calling is the process of identifying such variants for the sample under consideration. These variants can eventually throw light on many structural and functional anomalies embedded in the genomes and its repeat regions, manifesting in the form of structural and Copy Number Variations (CNVs), Single Nucleotide Polymorphisms (SNPs) etc. A precise alignment achieved through ReneGENE-GI's SRM enables a variant calling of high quality and confidence levels, allowing a more precise genotyping and phenotyping in presence of fusion genes and translocations within repeat regions in a genome. The SRM output from ReneGENE-GI is presented to variant calling tools like GATK, SAMTOOLS, FreeBayes etc which provide the resultant variant calls in the standard VCF format.

Amidst a wide variety of state-of-the art GI solutions [28, 29], the genomic computing community faces a lack of consensus or standards in brewing a flawless elucidation of biologically relevant information. In addition, the choices of algorithms and implementations in intermediate stages of GI have been subjective enough to snub out the useful information for downstream analyses, in the process of optimizing and accelerating the pipeline. As a result, downstream analyses continue to suffer due to the sufficiently large heuristics-driven errors that creep into the pipeline and subsequent biologically relevant inferences. In this context, the ReneGENE-GI pipeline stands out in offering the optimal choice for performing GI, over a fully accelerated pipeline, with an underlying confidence in the biologically significant and causative inferences made downstream.

4 ReneGENE CGM: The Comparative Genomics Module of ReneGENE-GI

The ReneGENE-CGM runs on accelerator platforms like FPGAs and GPUs. We have two flavours of this module, ReneGENE-AccuRA for FPGAs and ReneGENE-GMAccS for GPUs.

4.1 AccuRA: The SRM Pipeline for FPGAs

ReneGENE-GI's CGM is implemented on a reconfigurable accelerator platform as ReneGENE-AccuRA. This is an extended version of AccuRA, published in our earlier work [30, 31], which presents AccuRA's architecture, algorithms, mathematical model and scalability analysis. The AccuRA hardware archetype is presented in Fig. 4.

The SRM performed by the CGM, when applied to very long genomic sequences, is interpreted as an Approximate String Matching (ASM) problem. SRM algorithmically analyses the structural, functional and evolutionary relationship between the two input strings. SRM attempts to search the specific short read string q of length —q— (ranging from about 25 to a few hundred bases), over a much longer reference genome string G of

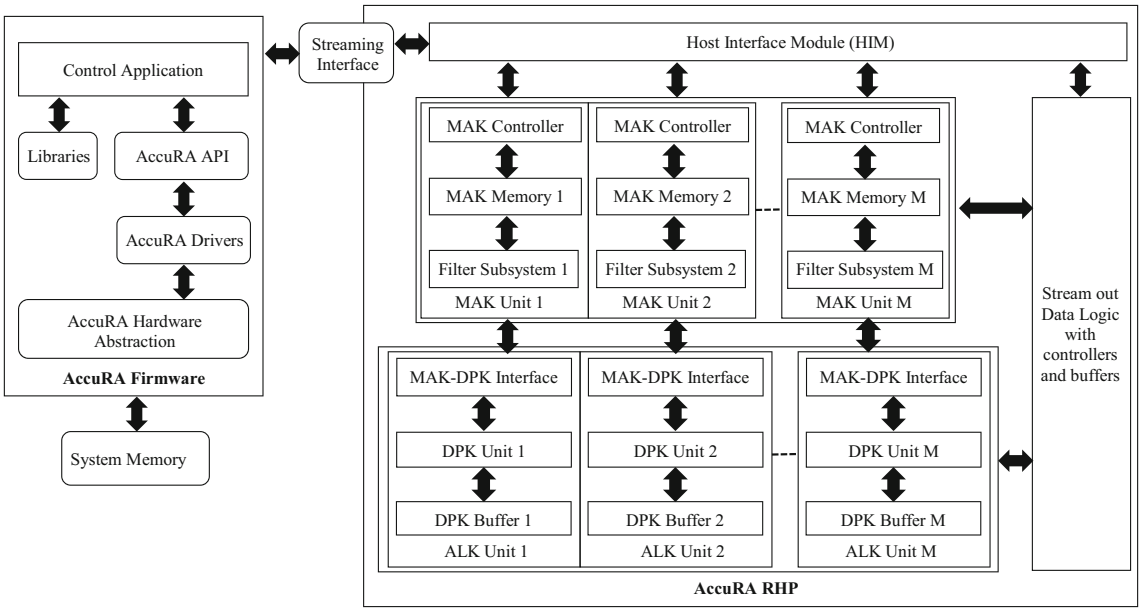

Figure 4 AccuRA SRM Pipeline Architecture: The RHP hosts Mapper Kernel (MAK) Units embedded with filter subsystem and Aligner Kernel (ALK) Units embedded with Dynamic Programming Kernel (DPK) Units.

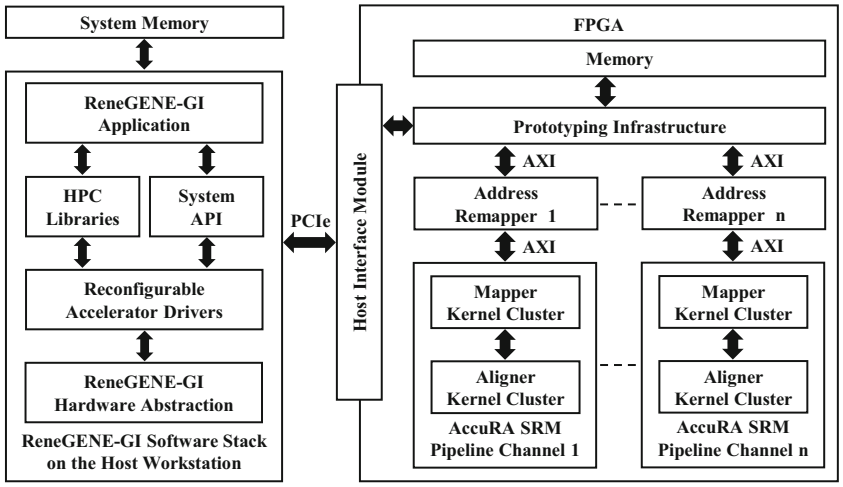

Figure 5 ReneGENE-AccuRA: The multi-channel architecture based on AccuRA SRM pipeline.

length —*G*— (a human reference genome is typically 3 billion bases long). The aim is to find the regions of origin of each short read string with respect to the reference, and hence find regions of similarity or dissimilarity, over the character set $\Sigma = \{A, C, G, T\}$.

The Dynamic Programming Kernel (DPK) units in AccuRA's hardware host a highly efficient and parallel DPK kernel to achieve traceback in hardware, based on a DP alignment algorithm as seen in Algorithm 3. The hardware performs alignment, in the shortest deterministic time, agnostic to short read length. AccuRA achieves a significant improvement in performance over conventional RHP models for SRM, with adequate sequence partitioning and scheduling schemes in the SRM workflow. By performing traceback in hardware overlapped with the forward scan during alignment, AccuRA eliminates the memory bottleneck issues and reduces the compute intensive tasks on the host significantly. The AccuRA prototype, configured on a reconfigurable hardware like FPGA, scaled well towards accommodating the big data of short reads of varying lengths, from smaller prokaryotic genomes to the larger mammalian genome, with a fine-grained single nucleotide resolution.

4.2 ReneGENE-AccuRA: A Multichannel Implementation of AccuRA SRM Pipeline on FPGAs

The scalability analysis and results from various prototypes in our earlier work proved to substantiate the scalability and performance of the parallel AccuRA SRM pipeline, making it a promising target to accelerate the SRM process in the NGS pipeline. Here, we present ReneGENE-AccuRA, a multi-channel, scalable and massively parallel computing pipeline that performs ultra-fast alignment of DNA short reads, presented in Fig. 5. Each channel of ReneGENE-AccuRA is composed of one AccuRA SRM pipeline, hosting several DPK and mapper units. A single reconfigurable hardware like FPGA can host multiple such AccuRA SRM pipelines. Supplemented with multi-threaded firmware architecture, ReneGENE-AccuRA precisely aligns short reads, at a fine-grained single nucleotide resolution, and offers full alignment coverage of the genome including repeat regions. ReneGENE-AccuRA is a fully streaming solution that eliminates memory bottleneck and storage issues, thus reducing the computing and I/O burden on the host significantly. With an appropriate data streaming pipeline, we provide an affordable solution, customizable according to scalability needs and budget availability. It is also pluggable to any genome analysis pipeline for use across multiple domains from research to clinical environment.

4.3 ReneGENE-GMAccS: A Multichannel Implementation of SRM on GPUs

ReneGENE-GMAccS, presented in Fig. 6 is a scalable, massively parallel and heterogeneous GPU-based model for SRM. This is a heterogeneous Single Instruction Multiple Data (SIMD) system, for accelerating SRM process in the NGS pipeline. This architecture implements Algorithm 3 for ReneGENE-GI across multiple parallel computational threads on GPUs.

ReneGENE-GMAccS efficiently handles the task-level parallelism and data level parallelism that is implicit within the ASM problem presented. The Rene-GENE-GMAccS firmware runs on a multi-node multi-core host platform, hosting several kernels in a pipelined fashion. These kernels are scheduled to run on a GPU based Accelerator Platform (GAP) housing single or multiple GPUs. The firmware allows for dynamic balancing of computations and flexible memory hierarchy. Each kernel performs a complex, coarse grained to fine grained parallel task, executed on a

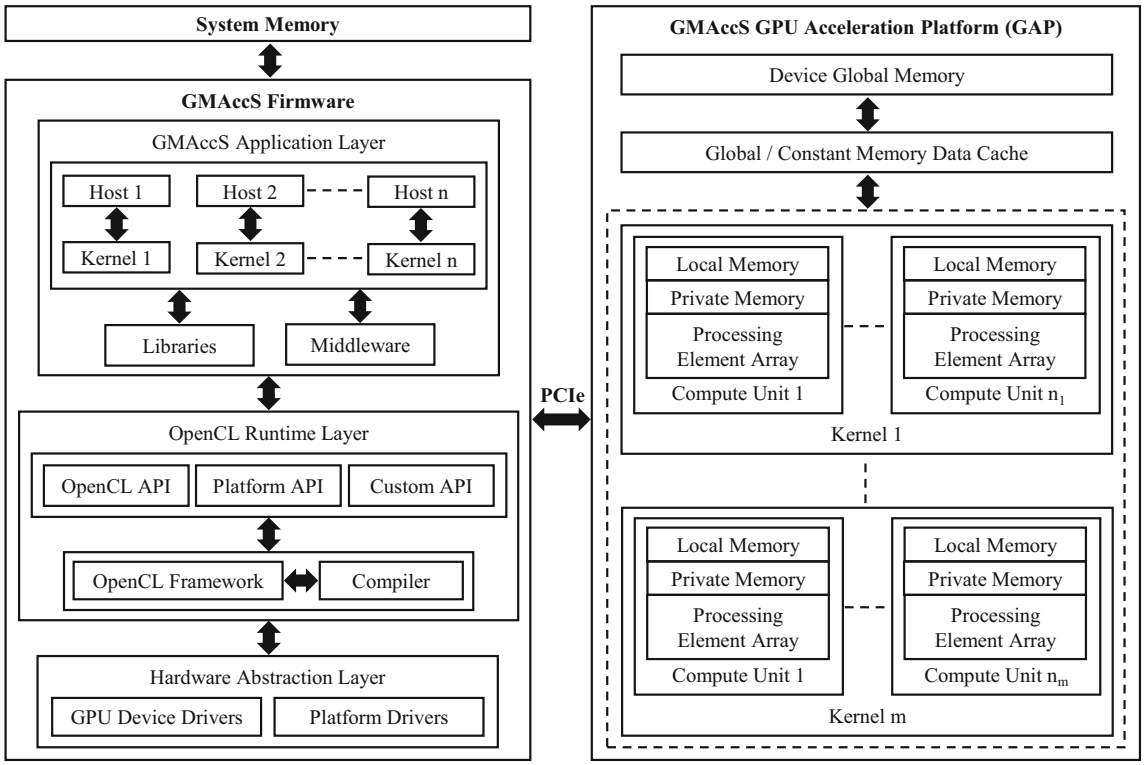

Figure 6 The ReneGENE-GMAccS architecture.

collection of data elements. With a seamless streaming of data between the host and the GAP, ReneGENE-GMAccS presents a massively parallel HPC model for SRM.

4.4 ReneGENE-GMAccS OpenCL Kernel Model on GPUs

We have implemented SRM algorithms in ReneGENE-GMAccS as kernels written in Open Computing Language (OpenCL), which makes it platform independent. It also enables ReneGENE-GMAccS to run on a heterogeneous parallel computing environment, hosting GPUs from multiple vendors. OpenCL also offers Just-In-Time (JIT) compile options, which allows the end-to-end application to make a superior use of the target GPU platform. ReneGENE-GMAccS has leveraged the portability of OpenCL and the compute efficiency of GPUs, through simplified wrappers and libraries. The kernels take advantage of the concurrent and parallel programming model provided by the OpenCL framework, in association with the application code developed in C/C++ that resides on the host.

The OpenCL platform model comes with an inbuilt host management layer and device side language support, and defines the relationship between the Rene-GENE-GMAccS host and the GAP. Each device on the GAP is an abstraction of a set of Compute Units (CUs), with each CU hosting a set of Processing Elements (PEs), presenting a Single Instruction Multiple Thread (SIMT) parallel execution model.

A scalable execution model is realized, while deploying fine-grained work items or batches of short read inputs onto the GAP. The kernels define the number of work-items as an n-dimensional range or NDRange. The work items are deployed as fixed size work groups across CUs in the Shared Multiprocessors (SM) of the GPU. The workgroups get scheduled as a group of threads on a set of PEs.

Each schedulable group of threads is termed *wavefront* on the hardware, where all the threads execute the same instruction while being on different control paths. Several such work groups run concurrently in a large batch of execution. An appropriate choice of NDRange and workgroup size can result in the most suitable occupancy levels for the GAP. Occupancy, a measure of concurrency within the GAP, decides the system performance while streaming large batches of short reads.

ReneGENE-GMAccS follows a fully consistent OpenCL shared memory model. The kernels allow synchronized host-device and device-host data transfers, while running ReneGENE-GMAccS at multiple levels of granularity. This helps the ReneGENE-GMAccS firmware to efficiently interleave periods of computations and communication, while handling ASM on large batches of short reads. By

Table 2 ReneGENE-GMAccS prototype platform details.

Feature	Single-GPU: P1	Multi-GPU: P2 (on Cray XC40)
Application environment	C++	C++
Kernel environment	STDCL/OpenCL	OpenCL
Middleware	–	MPI
Host processor	8-core 3.5 GHz AMD FX™-8320	12-core 2.4 GHz Intel Xeon
System memory	32 GB	64 GB per node
GPU hardware	Nvidia GeForce GTX 780 Ti	Nvidia Tesla K40
GPU processing units	2880	$2880 \times 24 = 69120$
GPU base clock rate	875 MHz	745 MHz
GPU memory	3 GB	12 GB $\times$ 24
GPU memory bandwidth (GB/s)	336	288
Compute capability	3.5	3.5

choosing the appropriate batch size, NDRange and level of granularity, ReneGENE-GMAccS OpenCL kernels exploit the underlying GPU architecture-level parallelism to its full potential.

5 ReneGENE-GI: Solutions and Results

Here, we present the details of the prototypes developed for ReneGENE-AccuRA and ReneGENE-GMAccS. The performance comparison for the above CGMs are provided for the SRM conducted on short reads for the human genome data set.

5.1 ReneGENE-GI: Solutions

5.1.1 ReneGENE-GI Host Environment

The software stack, that runs on the host, comprises of the preprocessing and post-processing modules of the ReneGENE-GI pipeline. This includes: (i) the reference index hashing step based on the MMPH algorithm, (ii) the read-lookup algorithm against the indexed reference for candidate genomic locations for a probable alignment, (iii) the HPC platform specific libraries and middleware, (iv) the hardware abstraction layer with the corresponding device drivers and platform drivers, (v) the post-processing module that makes decisions for the best alignment, secondary alignments, the corresponding computations for alignment/map qualities and (vi) a subsequent formatting of the output data in the Sequence Alignment (SAM) format. The pipeline also allows conversion of the SAM file to its compressed Binary Alignment (BAM) file and its verification towards fitness for downstream NGS data analytics.

5.1.2 Prototype Model for ReneGENE-GMAccS

To evaluate ReneGENE-GMAccS for its performance and scalability, we have developed a prototype model based on both single and multiple GPUs. The single GPU environment (Platform P1) is a workstation, hosting an 8 core AMD processor coupled with a Nvidia GPU. To evaluate the scalability features of ReneGENE-GMAccS, we modeled the same on SahasraT, the Cray XC40 based in-house supercomputing cluster, with upto 24 GPUs put to use for alignment in parallel (Platform P2) [32]. The prototype has a set of three kernels, embedded in a buffered pipeline. The details of the platforms are shown in Table 2.

5.1.3 Prototype Model for ReneGENE-AccuRA

ReneGENE-AccuRA was prototyped on an HPC platform supported with a reconfigurable accelerator card built on multiple Xilinx Virtex 7 XC7V2000T devices, that is scalable upto 633 million ASIC gates. The host interface is through a Kintex-7 XC7K325T-FBG-900 FPGA. The host processor is interfaced to the Kintex-7 FPGA via the high speed interface of PCI-E x8 gen3. The embarassingly parallel bio-computing in AccuRA's SRM is further favoured by the inherent reprogrammability of FPGAs, massively parallel compute resources, extreme data path parallelism and fine grained control mechanisms offered by the FPGAs.

5.1.4 ReneGENE-AccuRA Hardware

The multi-channel ReneGENE-AccuRA is represented as DUT within each FPGA. It is interfaced with the prototyping infrastructure on the FPGA through the

Table 3 Scalability analysis parameters.

Symbols	Description
BW_{in}	Streaming input bandwidth
L	Short read length
B	Streaming buffer depth
l	Subsequence length for short read
m	Number of partitions for input short read, with overlapping
b	Number of bits for encoding each base of input short read
bl	Streaming buffer width
τ_{MAK}	MAK unit clock period
τ_{DPK}	DPK unit clock period
x	MAK unit operating cycles
y	DPK unit operating cycles
P	Total number of pairs launched on MAK-DPK units for SRM
N	Number of MAK-DPK units deployed on a single AccuRA SRM pipeline channel
$p = \frac{P}{N}$	Number of pairs allotted for SRM, per MAK-DPK unit
C	No. of cell updates per DPK Unit
K	No. of filter kernel operations per MAK Unit
R_{in}	No. of filter kernel operations per MAK Unit $R_{in} = \frac{8 \times BW_{in}}{b \times m \times l}$
T_{MAK}	Total MAK unit time to cover P pairs $T_{MAK} = x \times \tau_{MAK}$
T_{DPK}	Total DPK unit time to cover P pairs $T_{DPK} = y \times \tau_{DPK}$
R_{RHP}	Read Processing Rate of a single AccuRA SRM pipeline channel $R_{RHP} = \frac{N}{T_{MAK}+T_{DPK}}$
T_{RHP}	Alignment Time of a single AccuRA SRM pipeline channel $T_{RHP} = p \times (T_{MAK} + T_{DPK})$
$T_{single\_channel\_AccuRA}$	The total time invested in performing SRM in a single AccuRA SRM pipeline channel $T_{single\_channel\_AccuRA} \approx T_{Load} + T_{RHP} + T_{Unload}$
P_{MAK}	Performance of MAK units within a single AccuRA SRM pipeline, measured in terms of Giga Maps Per Second (GMPS) $P_{MAK} = \frac{N \times K}{x \times T_{MAK}}$
P_{DPK}	Performance of DPK units within a single AccuRA SRM pipeline, measured in terms of Giga Cell Updates Per Second (GCUPS) $P_{DPK} = \frac{N \times C}{y \times T_{DPK}}$

standard AXI4 interface with 256 bit-wide data bus, running at a frequency of 125MHz. The Address Remapper unit allows an automatic remapping of the address spaces of DUT for transactions, allowing an ease of scalability in adding more AccuRA SRM channels to the DUT. The implementation is done using VHDL and Verilog.

5.1.5 Scalability Analysis for ReneGENE-AccuRA

The parameters in scalability analysis for ReneGENE-AccuRA are given in Table 3. Consider the multi-channel AccuRA SRM pipeline model, where reads are streamed in at the rate R_{in} (measured as Giga Reads/second or GR/s) over an input streaming bandwidth of BW_{in} (measured as Giga Bytes/second or GB/s). The m subsequences of short reads, each of length l, are streamed through a streaming buffer of depth B, which holds one subsequence in each word of storage.

Each MAK unit performs filtering in time T_{MAK}, over x cycles of the MAK unit clock, with period τ_{MAK}. Each DPK unit performs alignment in time T_{DPK}, over y cycles of the DPK unit clock, with period τ_{DPK}. If N MAK-DPK units are configured within a single AccuRA SRM pipeline channel, then each unit gets its share of p pairs for performing SRM. The single AccuRA SRM pipeline channel thus performs N SRMs in a total time of $T_{MAK} + T_{DPK}$, with N MAK-DPK units running in parallel. The single channel hence processes reads at a rate of R_{RHP} measured in GR/s. At this rate, the hardware aligns all the P reads, with p reads aligned in parallel over N MAK-DPK units, over a total time of T_{RHP}.

For scaling up the performance, let us include C such channels of AccuRA SRM pipelines within a single FPGA. Here, each channel will take the same amount of time to process the same number of reads.

Now, the overall performance from all the MAK units from C channels, measured in terms of Giga Maps Per Second (GMPS), is given by:

$$P_{MAK} = \frac{C \times N \times K}{x \times T_{MAK}} \tag{1}$$

Table 4 Small genome data.

Genome	Reference length	Read size	Number of reads
E.coli (E.c)	4641652	150	1374751
P.aeruginosa (P.a)	6264404	300	1245456
S.cerevisiae (S.c)	12157105	50	11975806
S.pombe (S.p)	12591251	101	2467348
M.tuberculosis (M.tb)	4411532	50	4872186
H.Infuenzae (H.in)	1830092	101	1781164
B.pertussis (B.p)	4086189	101	1794963

The overall performance of DPK unit, measured in terms of Giga Cell Updates Per Second (GCUPS), is given by:

$$P_{DPK} = \frac{C \times N \times C}{y \times T_{DPK}} \quad (2)$$

Thus, we see that by scaling up the single AccuRA SRM pipeline channel, by increasing N, the ReneGENE-AccuRA hardware gains a better throughput, as it can handle more pairs in parallel. The scalability is complemented by further scaling up the number of such channels, C, within a single FPGA. The number of such channels within an FPGA is limited only by the allowed reconfigurable hardware space for the DUT within the FPGA. The input data is then fairly divided among the channels, so that the SRM process is complete in approximately $1/C$ times the total time taken for SRM by a single channel.

5.2 ReneGENE-GI: Results

5.2.1 Comparing ReneGENE-GI CGM with Existing Aligners

Here, we compare the basic multi-core implementation of ReneGENE-GI's CGM with the Open source distributions of widely used aligners namely, BWA-MEM and Bowtie2, without the support of any acceleration hardware.

We have selected the short read data for a list of small organisms, listed in Table 4, for running the Rene-GENE-GI pipeline. These data vary in the length of the reads and the reference. The performance comparisons are listed in Table 5. From the table, we can see that for these organisms, ReneGENE-GI is much faster than the other two SRM tools. In the subsequent sections, we present the results for ReneGENE-GI on accelerator platforms with larger genomes.

Table 5 ReneGENE-GI Versus state-of-the-art SRM comparisons (time taken in seconds).

Genome	Bowtie2	BWA-MEM	ReneGENE-GI
E.coli (e.c)	325.1	282.8	43.28
P.aeruginosa (p.a)	1278.41	471.49	104.22
S.cerevisiae (s.c)	2778.1	845.3	436.2
S.pombe (s.p)	1032.34	304.5	107.52
M.tuberculosis (m.tb)	773.33	307.82	146.2
H.influenzae (h.in)	188.57	248.75	52.72
B.pertussis (b.p)	3934.4	1714.7	165.2

Table 6 ReneGENE-GI variant calling comparison with alignment to forward reference strand: results derived from the output of comparative genomics module of the ReneGENE-GI pipeline.

Organism	Bowtie 2.0	BWA-MEM	ReneGENE-GI
1. b.p	60	487	582
2. s.c	33804	34348	65758
3. s.p	207	250	725
4. m.tb	448	444	1114

ReneGENE-GI has reported an increased number of valid alignments with the precision and accuracy of reference locations. This has helped in reporting several unique variants (changes in genomic alphabets) at specific genomic locations, as part of the alignment process. Through a downstream process called variant calling, the SRM output was analysed and scanned for quality and quantity of such variants. A comparison of the variants derived from ReneGENE-GI and those derived from Bowtie2 and BWA-MEM was done, the details of which are captured in Table 6. The table shows that several variants are exclusively reported by ReneGENE-GI, which would be relevant while looking for structural and biological interpretations and filtering out causative and actionable variants, which otherwise would have been purged by the other SRM tools.

5.2.2 Performance Evaluation of ReneGENE-GI for Large Genome Benchmarks

The ReneGENE-GI prototypes for FPGA and GPU platforms were tested by running SRM for very large data sets of the order of several Giga Bytes, for the mammalian human genome. The details of the input data set is provided in Table 7. We have used the GrCh38 reference genome assembly, which is around 3 billion bases long, consisting of 23 chromosomes and mitochondrial DNA. We have considered alignment of three human genomes, each of which corresponds to a family of father (SRR1559289, SRR1559290, SRR1559291, SRR1559292, SRR1559293), mother (SRR1559294, SRR1559295, SRR1559296, SRR1559297, SRR1559298) and their child (SRR1559281, SRR1559282, SRR1559283, SRR1559284). Here, each read is 200 bases long. The reads are subjected to lookup against the reference gen-ome index. Subsequently, they are sent for alignment on the FPGA and GPU by streaming over the PCIe link through buffers. The

Table 7 Human genome experiment details.

ID	SRR Read	No. of reads	No. of bases	Buffer contents for SRM	No. of streamed batches
1	SRR1559289	27594045	5.5G	1545583696	82
2	SRR1559290	28019239	5.6G	1567102768	84
3	SRR1559291	169777482	34G	9616084216	510
4	SRR1559292	168278483	33.7G	9031420804	479
5	SRR1559293	168484341	33.7G	9194814968	488
6	SRR1559294	180827103	36.2G	10449482200	554
7	SRR1559295	96741850	19.3G	5716247292	303
8	SRR1559296	148849161	29.8G	8719041172	462
9	SRR1559297	33028205	6.6G	1872052764	100
10	SRR1559298	33621893	6.7G	1899810824	101
11	SRR1559281	146929886	29.4G	8661172200	459
12	SRR1559282	143848074	28.8G	8348191016	443
13	SRR1559283	144871968	29G	8415342112	446
14	SRR1559284	142831237	28.6G	8303472652	440

Table 8 ReneGENE-GMAccS alignment time per chromosome for human genome read sets on P1. The values indicate time taken in seconds.

Read →	Father, SRR15592xx					Mother, SRR15592xx					Child, SRR15592xx			
Chr ↓	89	90	91	92	93	94	95	96	97	98	81	82	83	84
1	24.61	26.3	162.66	149.16	143.99	160.49	88.38	128.91	29.3	28	144.05	140.48	132.93	137.5
2	19.61	21.06	129.7	119.06	114.99	126.11	70.7	102.58	23.04	22	116.12	112.79	106.75	110.6
3	17.62	18.86	116.6	106.37	103.04	117.04	65.41	94.97	21.44	20.45	103.34	100.43	95.04	98.44
4	16.85	18	111.25	101.55	98.15	105.91	59.16	86.3	19.5	18.69	97.74	95.1	90.24	93.34
5	19.33	20.65	127.6	116.88	112.82	126.63	69.12	101.52	23.29	22.27	112.39	109.77	103.91	107.48
6	14.71	15.72	97.21	88.67	86.01	91.65	51.45	74.56	16.73	15.94	90.03	87.36	82.74	85.62
7	27.69	29.81	184.06	165.85	162.25	181.96	100.21	147.9	33.68	32.11	169.86	166.95	157.35	163.14
8	13.83	14.82	91.74	83.35	80.85	86.7	48.3	70.4	15.86	15.17	80.29	78.18	74.07	76.76
9	15.1	16.08	99.4	90.58	87.96	93.63	51.75	75.73	17.29	16.49	86.08	84.23	79.79	82.61
10	12.19	13.08	80.6	73.91	71.56	77.94	43.6	63.18	14.2	13.59	70.87	68.95	65.24	67.51
11	22.42	23.97	147.89	133.66	129.33	137.13	74.7	109.88	25.12	23.87	113.53	110.96	105.14	108.41
12	17.99	19.36	118.66	108.41	104.86	108.73	59.86	87.68	20.03	19.03	97.59	94.78	90.01	93.43
13	9.35	9.95	62.01	56.39	54.64	59.39	32.72	47.78	10.83	10.38	56.27	54.86	52.03	53.9
14	10.33	10.99	68.45	62.39	60.42	65.8	36.38	53	11.99	11.49	62.08	60.49	57.34	59.42
15	9.85	10.55	65.1	59.35	57.43	64.98	35.84	52.51	11.87	11.37	63.87	62.81	59.23	61.29
16	12.25	13.22	80.86	74.05	71.46	76.36	41.78	61.32	13.97	13.41	73.38	71.66	67.73	70.15
17	11.49	12.3	75.69	69.39	67.28	90.92	50.25	73.01	16.44	15.63	76.11	74.04	70.09	72.56
18	19.29	20.59	129.65	116.16	113.04	104.97	56.91	84.91	19.52	18.86	109.86	107.61	102.75	106.77
19	12.99	13.87	85.69	79.06	76.1	85.86	46.28	68.03	15.72	15.04	74.71	73.24	69.27	71.66
20	11.01	11.8	73.02	65.63	63.95	60.62	33.27	48.95	11.19	10.72	62.97	61.65	58.36	60.42
21	5.95	6.32	39.54	35.82	34.73	36.96	20	29.46	6.75	6.49	35.81	35.06	33.28	34.5
22	7.01	7.45	46.49	42.35	41.01	44.12	24.12	35.24	8.02	7.69	42.28	41.31	39.17	40.62
X	17.24	18.4	113.72	103.7	100.18	160.06	88.41	130.16	29.64	28.06	122.72	119.55	113.13	116.76
Y	2.31	2.46	15.22	13.8	13.39	12.57	7.03	10.23	2.31	2.21	13.61	13.22	12.52	12.95
MT	0.007	0.008	0.049	0.042	0.041	0.038	0.025	0.035	0.007	0.007	0.046	0.043	0.041	0.043
Total (minutes)	5.85	6.26	38.71	35.26	34.16	37.94	20.93	30.64	6.96	6.65	34.59	33.76	31.97	33.1

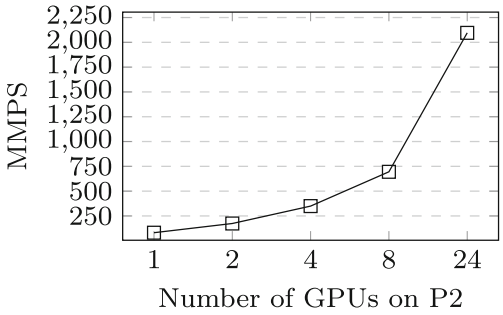

Figure 7 ReneGENE-GMAccS performance in MMPS, on P2 with multiple GPUs.

sample FPGA buffer sizes, are configured to hold up to 18874368 words of data in one batch, as shown in Table 7. For GPUs, this is configured across two variables, the batch size (number of parallel GPU compute threads) and the NDRange (NDR).

5.2.3 Results from Large Genome Benchmarks for ReneGENE-GMAccS on P1

Table 8 captures the time taken by P1, in seconds, to align the read sets across all the chromosomes in the reference. We can see that, for the largest read set SRR1559291, P1 could align about 169 million short reads of 100 bases each, in a total time of 162.66 seconds against chromosome 1. Thus, with a single GPU, we could achieve SRM for the entire read set of 169777482 reads, against all the chromosomes within the human reference genome, in about 38.71 minutes, which is the GPU run time for all the alignments.

Table 9 ReneGENE-GMAccS overall alignment time comparison for human genome read sets.

SRR Read Sets	P1 time (s)	P2 time 24 GPUs (s)
SRR1559289	351.02	17.18
SRR1559290	375.61	18.39
SRR1559291	2322.85	113.71
SRR1559292	2115.58	103.61
SRR1559293	2049.49	100.33
SRR1559294	2276.57	111.45
SRR1559295	1310.95	64.17
SRR1559296	1838.24	89.99
SRR1559297	417.75	20.45
SRR1559298	398.94	19.53
SRR1559281	2075.57	101.61
SRR1559282	2025.54	99.16
SRR1559283	1918.12	93.9
SRR1559284	1985.89	97.22

Table 10 ReneGENE-AccuRA utilization report, with single and dual channel AccuRA SRM pipeline single Xilinx Virtex 7 XC7V2000T device.

Feature	Single channel	Percentage utilization: single channel	Dual channel	Percentage utilization: dual channel
Number of Slice Registers	128708 out of 2443200	5.26	213366 out of 2443200	8.73
Number of Slice LUTs	170177 out of 1221600	13.93	310045 out of 1221600	25.35
Number of bonded IOBs	166 out of 850	19.52	96 out of 850	19.52
Number of Block RAM/FIFO	203 out of 1292	15.71	374 out of 1292	28.90

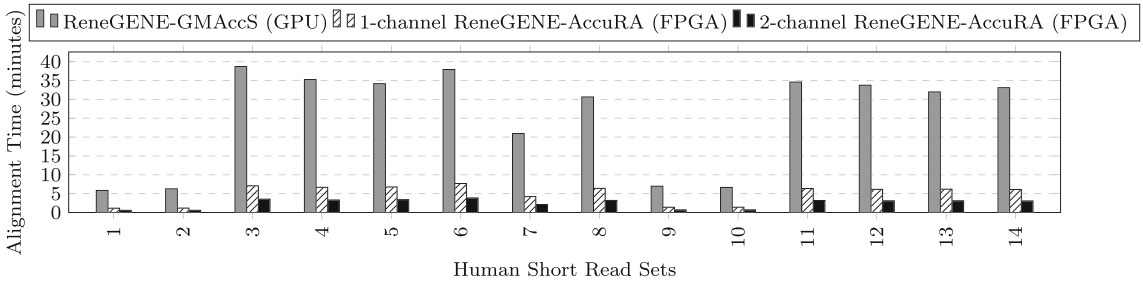

Figure 8 Performance comparison of FPGA versus GPU for human short read sets.

5.2.4 Results from Large Genome Benchmarks for ReneGENE-GMAccS on P2

We have exploited the scalability of ReneGENE-GMAccS to achieve a better SRM performance on P2, where the system was configured to use up to 24 GPUs in parallel.

Figure 7 shows the performance of P2, with 1, 2, 4, 8 and 24 GPUs, for a batch size of 1283776 and NDR of 64, for the large genome read sets. Table 9 depicts the total time taken by P1 and P2 to align all the input read sets. Here we can see that, with 24 GPUs, P2 took 113.71 seconds to complete the alignment of the largest read set SRR1559291, as against P1, which took 2322.85 seconds to finish the job with a single GPU. Thus, from Fig. 7 and Table 9, the performance improvement is evident in terms of the throughput levels measured in Million Maps Per Second (MMPS) as well as the total time taken. The distributed memory architecture on P2 along with the MPI middleware in ReneGENE-GMAccS thus helps in achieving a more-than-linear improvement in performance without overwhelming the shared resources, as the number of GPUs increases.

5.2.5 Results from Large Genome Benchmarks for ReneGENE-AccuRA

The ReneGENE-AccuRA prototype was tested with single and dual channel AccuRA SRM pipelines within a single FPGA while aligning the human short read sets. Each channel hosted 16 MAK units and 16 DPK units. With this configuration, to align 500 million reads (100 bases long) against the reference genome (3 billion bases long), with each read reporting a mapping at five locations on the reference, ReneGENE-AccuRA performs 4.65 Tera map operations and 10.24 Tera cell updates at the rate of 21.14 GMPS and 46.56 GCUPS in about 3.68 minutes. The implementation results for the dual-channel ReneGENE-AccuRA are provided in Table 10.

For the human genome read sets in Table 7, the alignment times for various configurations are shown in Fig. 8. Here, we can see that the time taken by ReneGENE-AccuRA is about one-fifth (with single channel AccuRA SRM pipeline) and about one-tenth (with dual channel AccuRA SRM pipeline), the time taken by the single GPU OpenCL implementation of ReneGENE-GI's CGM. This single GPU implementation is itself 2.62x faster than CUSHAW2-GPU (the GPU CUDA implementation of CUSHAW) [33, 34]. With the single-GPU implementation demonstrating a speedup of 150x over standard heuristic aligners in the market like BFAST [35], the reconfigurable accelerator version of ReneGENE-AccuRA is several orders faster than the competitors, offering precision over heuristics. By extending the implementation to four and six channels within a single FPGA, there is a definite increase expected in the performance as evident from the scalability analysis. With multiple FPGAs available on the platform, the scope for further improvement in performance increases with increase in number of FPGAs and number of channels supported within the FPGAs.

6 Conclusion

Through this paper, we have presented ReneGENE-GI, an innovatively engineered GI pipeline. The pipeline strikes the right balance between comparative genomics and *de novo* read extension, to run an irregular application like GI. With parallel algorithms executed on reconfigurable accelerator hardware, ReneGENE-GI exploits the inherent parallelism and scalability of the hardware at the level of micro and system architecture, amidst fine-grain synchronization.

The k-mer based dynamic MMPH algorithm for reference genome indexing provides an accurate hash table, allowing a heuristic free multi-read alignment across repeat regions of the reference. Supplemented with a multi-threaded firmware architecture, the CGM in ReneGENE-GI precisely aligns short reads, at a fine-grained single nucleotide resolution, and offers full alignment coverage of the genome including repeat regions. The CGM has been deployed on two accelerator platforms, as ReneGENE-AccuRA and ReneGENE-GMAccS, on FPGA and GPU respectively, The parallel dynamic programming kernels on multiple channels of CGM seamlessly perform

traceback process in hardware in parallel with forward scan, thus achieving short read mapping in a minimum possible deterministic time.

ReneGENE-GI is a fully streaming solution that eliminates memory bottleneck and storage issues, thus reducing the computing and I/O burden on the host significantly. The performance analysis shows that Rene-GENE-AccuRA is faster in comparison with ReneGENE-GMAccS and the state-of-the-art aligners, with similar levels of precision and accuracy, while aligning significantly large volumes of human genome data. With an appropriate data streaming pipeline, we provide an affordable solution, customizable according to scalability needs and budget availability. It is also pluggable to any genome analysis pipeline for use across multiple domains from research to clinical environment. The precise secondary analysis offered by the CGM of ReneGENE-GI running on accelerator hardware, associated with an efficient tertiary analysis down- stream serves to be a promising target to derive more meaningful inferences from NGS data with biological and clinical significance.

References

1. Frese, K.S., Katus, H.A., Meder, B. (2013). Next-generation sequencing: from understanding biology to personalized medicine. *Biology*, *2*(4), 378–398.
2. Mardis, E.R. (2011). A decade's perspective on dna sequencing technology. *Nature Perspective*, *470*, 198–203.
3. Stephens, Z.D., Lee, S.Y., Faghri, F., Campbell, R.H., Zhai, C., Efron, M.J., et al. (2015). Big data: Astronomical or genomical? PLOS Biology, 13(7).
4. Navarro, G. (2001). A guided tour to approximate string matching. *ACM Computing Surveys*, *33*(1), 31–88.
5. Aho, A.V., & Corasick, M.J. (2000). Efficient string matching: an aid to bibliographic search. *IEEE Data Engineering Bulletin*, *24*(4), 19–27.
6. Costa, F.F. (2012). Big data in genomics: Challenges and solutions. *G.I.T Laboratory Journal*, *11*(12), 2–4.
7. Marx, V. (2013). The big challenges of big data. *Nature*, *498*, 255–260.
8. Reinert, K., Langmead, B., Weese, D., Evers, D.J. (2015). Alignment of Next-Generation Sequencing Reads Annu. Rev Genomics Hum. Genet., 133–151.
9. Baker, M. (2010). Next-generation sequencing: adjusting to data overload. *Nature Methods*, *7*, 495–499.
10. Treangen, T.J., & Salzberg, S.L. (2012). Repetitive dna and next-generation sequencing: computational challenges and solutions. *Nature Reviews*, *13*, 36–46.
11. Flicek, P., & Birney, E. (2009). Sense from sequence reads: methods for alignment and assembly. *Nature Methods*, *6*, S6–S12.
12. Yamaguchi, Y., Maruyama, T., Konagaya, A. (2002). High speed homology search with FPGAs. In Proceedings of the Pacific Symposium on Biocomputing (pp. 271–282).
13. Benkrid, K., Liu, Y., Benkrid, A. (2009). A highly parameterized and efficient FPGA-based skeleton for pairwise biological sequence alignment. *IEEE Transactions On Very Large Scale Integration Systems*, *17*(4), 561–570.
14. Razmyslovich, D., Marcus, G., Gipp, M., Zapatka, M., Szillus, A. (2010). Implementation of Smith-Waterman Algorithm in openCL for GPUs. In IEEE Second International Workshop on High Performance Computational Systems Biology (pp. 48–56).
15. Banerjee, S.S., El-Hadedy, M., Lim, J.B., Kalbarczyk, Z.T., Chen, D., Lumetta, S.S., Iyer, R.K. ASAP: Accelerated Short-Read Alignment on Programmable Hardware.
16. Ergin, M.A., Hassan, H., Xin, H., Alli, E. (2017). Gatekeeper: a new hardware architecture for accelerating pre-alignment in DNA short read mapping. Bioinformatics.
17. Arram, J., Kaplan, T., Luk, W., Jiang, P. (2017). Leveraging FPGAs for accelerating short read alignment. IEEE/ACM TRANSACTIONS ON COMPUTATIONAL BIOLOGY AND BIOINFORMATICS, VOL. 14, NO. 3.
18. Lee, C.Y., Chiu, Y.C., Wang, L.B., al et. (2013). Common applications of next-generation sequencing technologies in genomic research. *Translational Cancer Research*, *2*(1), 33–45.
19. Alyass, A., Turcotte, M., Meyre, D. (2015). From big data analysis to personalized medicine for all: challenges and opportunities. BMC Medical Genomics, 8(33).
20. Chen, C., & Schmidt, B. (2004). Performance analysis of computational biology applications on hierarchical grid systems. In Proceedings of the IEEE International Symposium on Cluster Computing and the Grid, CCGrid 2004 (pp 426–433). Chicago.
21. Bader, D.A. (2005). High-performance algorithm engineering for large-scale graph problems and computational biology. In Proceedings of the International Workshop on Experimental and Efficient Algorithms, WEA 2005 (pp. 16–21). Springer.
22. Natarajan, S., KrishnaKumar, N., Pal, D., Nandy, S.K. (2018). ReneGENE-GI: empowering precision genomics with FPGAs on HPCs. In Proceedings of the 14th International Symposium on Applied Reconfigurable Computing (ARC).
23. Myers, E. (1994). A sublinear algorithm for approximate keyword searching. *Algorithmica*, *12*, 345–374.
24. Smith, T.F., & Waterman, M.S. (1981). Identification of common molecular subsequences. *J. Mol Bwl.*, *147*, 195–197.
25. Altschul, S.F., Bundschuh, R., Olsen, R., Hwa, T. (2001). The estimation of statistical parameters for local alignment score distributions. *Nucleic Acids Research*, *29*, 351–361.
26. Natarajan, S., KrishnaKumar, N., Pavan, M., Pal, D., Nandy, S.K. (2018). ReneGENE-DP: accelerated parallel dynamic programming for genome informatics. In Proceedings of 2018 International Conference on Electronics, Computing and Communication Technologies (IEEE CONECCT).
27. Natarajan, S., KrishnaKumar, N., Anuchan, H.V., Pal, D., Nandy, S.K. (2018). ReneGENE-novo: co-designed algorithm-architecture for accelerated preprocessing and assembly of genomic short reads. In Proceedings of the 14th International Symposium on Applied Reconfigurable Computing (ARC).
28. Li, H., & Homer, N. (2010). A survey of sequence alignment algorithms for next-generation sequencing. *Briefings in Bioinformatics*, *2*, 473–483.
29. Hatem, A., Bozdag, D., Toland, A.E., Catalyurek, U.V. (2013). Benchmarking short sequence mapping tools. BMC Bioinformatics, 14.
30. Natarajan, S., KrishnaKumar, N., Pal, D., Nandy, S.K. (2016). AccuRA: accurate alignment of short reads on scalable reconfigurable accelerators. In Proc. IEEE International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS XVI) (pp. 79–87).
31. Natarajan, S., KrishnaKumar, N., Pal, D., Nandy, S.K. Accurate and accelerated secondary analysis of genomes: Implications for Genomics, NGS'17: Structural Variation and Population Genomics.

32. SERC, Indian Institute of Science, Bangalore. Sahasrat (Cray XC40). http://www.serc.iisc.in/facilities/cray-xc40-named-as-sahasrat.
33. Liu, Y., Schmidt, B., Maskell, D.L. (2012). CUSHAW: A CUDA compatible short read aligner to large genomes based on the Burrows-Wheeler transform. *Bioinformatics*, *28*(14), 1830–1837.
34. Liu, Y., & Schmidt, B. (2014). CUSHAW2-GPU: Empowering Faster gapped Short-Read alignment using GPU computing. *IEEE Design and Test of Computers*, *31*(1), 31–39.
35. Homer, N., Merriman, B., Nelson, S.F. (2009). BFAST: An alignment tool for large scale genome resequencing. PLoS 4.

Santhi Natarajan received the BE degree in Electronics and Communications Engineerging in 2001 from University of Kerala, India and MS degree in ASIC Design in 2010 from Mangalore University, India. She was formerly working as a scientist in Indian Space Research Organization (ISRO), and later as FPGA Design Engineer with Network Sound Inc, California, USA, and later as Technology Analyst with Infosys, Bangalore, India. She received her PhD Degree from the Centre for Nano Science and Engineering, Indian Institute of science, Bangalore, India, in 2017. Her research interests are high performance computing for bioinformatic applications, embedded computing, OpenCL.

Krishna Kumar N. received the BE and MTech degrees in Computer Science in 2010 and 2012 respectively from Visvesvaraya Technological University, Belgaum, India. He is currently a Senior Research Fellow at Biomolecular Computation Lab, in the Department of Computational and Data Sciences, Indian Institute of Science, Bangalore, India. His research interests include parallel programming, GPGPU programming, High Performance Computing and embedded systems.

Debnath Pal is a Professor in the Department of Computational and Data Sciences of the Indian Institute of Science, Bangalore. His research interests are in the areas of Computational Biology, Bioinformatics, Omics and Systems Biology. He received all his educational degrees in Chemistry with BSc (Hons.) from University of Kolkata in the year 1993, MSc from the Indian Institute of Technology, Kharagpur in 1995, and PhD from Jadavpur University, Kolkata, in 2000. He has over 50 research publications till date. He is fellow and member of several academic and professional bodies. He is noted for his contributions in the area of protein structure, function, interaction and dynamics. His research group regularly contributes to open access tools in biology.

S. K. Nandy is a Professor in the Department of Computational and Data Sciences of the Indian Institute of Science, Bangalore. His research interests are in areas of High Performance Embedded Systems on a Chip, VLSI architectures for Reconfigurable Systems on Chip, and Architectures and Compiling Techniques for Heterogeneous Many Core Systems. Nandy received the BSc (Hons.) Physics degree from the Indian Institute of Technology, Kharagpur, India, in 1977. He obtained the BE (Hons.) degree in Electronics and Communication in 1980, MSc (Engg.) degree in Computer Science and Engineering in 1986, and the PhD degree in Computer Science and Engineering in 1989 from the Indian Institute of Science, Bangalore. He has over 170 publications in International Journals, and Proceedings of International Conferences, and 5 patents. He is a valued IEEE member for 22 years.