REPUTE: An OpenCL based Read Mapping Tool for Embedded Genomics

Sidharth Maheshwari[†], Rishad Shafik[†], Ian Wilson[‡], Alex Yakovlev[†] and Amit Acharyya[◊]
[†]School of Engineering, Newcastle University, Newcastle upon Tyne, UK.
[‡]Institute of Genetic Medicine, Newcastle University, Newcastle upon Tyne, UK.
[◊]Department of Electrical Engineering, Indian Institute of Technology Hyderabad, Hyderabad, India.
E-mail: S.Maheshwari2@ncl.ac.uk, Rishad.Shafik@newcastle.ac.uk

***Abstract*—Genomics is transforming medicine from reactive to personalized, predictive, preventive and participatory (P4). The massive amount of data produced by genomics is a major challenge as it requires extensive computational capabilities, consuming large amounts of energy. A crucial prerequisite for computational genomics is genome assembly but the existing mapping tools used are predominantly software based, optimized for homogeneous high-performance systems. In this paper, we propose an OpenCL based REad maPper for heterogeneoUs sysTEms (REPUTE), which can use diverse and parallel compute and storage devices effectively. Core to this tool are dynamic programming based filtration and verification kernel to map the reads on multiple devices, concurrently. We show hardware/ software co-design and implementations of REPUTE across different platforms, and compare it with state-of-the-art mappers. We demonstrate the performance of mappers on two systems: 1) Intel CPU + 2×Nvidia GPUs; 2) HiKey970 embedded SoC with ARM Cortex-A73/A53 cores. The results show that REPUTE outperforms other read mappers in most cases producing up to 13× speedup with better or comparable accuracy. We also demonstrate that the embedded implementation can achieve up to 27× energy savings, enabling low-cost genomics.**

***Index Terms*—Genomics, read Mapping, embedded acceleration, heterogeneous computing, OpenCL.**

I. Introduction

Today, with over 6000 single-gene genetic conditions and many diseases involving genetic variants known, medicine has become the foremost application of genomics. It is a major driver in the undergoing transformation of medicine from reactive to a personalized, predictive, preventive and participatory (P4) form [1]. Genomics requires availability of genome for research, which can be obtained from the assembly pipeline of whole genome sequencing (WGS). Assembling genome of an organism can be done by mapping reads, which are small sections of genome obtained from sequencing machines, to the reference genome using read mapping tools [2]. State-of-the-art tools, such as [3]–[8], employ various data structures, approximate string matching algorithms and dynamic programming (DP) algorithms to identify the possible candidate locations of reads in the original genome, to re-assemble it.

Due to a large number of fragmented reads, genomics contributes to expansive data. Current trends indicate that our computation performance will scale poorly under the future demands generated from a large population [9]. A major reason for this is our insistence on hardware-agnostic, software based solutions. Besides performance, the energy required to process a large volume of data posits challenges from cost and environmental aspects [10].

Modern computing systems are heterogeneous and carry, at least, a GPU along with the CPU. Most state-of-the-art mapping tools [3]–[7], however, are optimized for CPU and are hardware oblivious. Several tools have been proposed for FPGA and GPU specific architecture, however, have not received widespread acceptability due to architecture specific implementations and other limitation imposed on the mapping parameters [11]. In [8], for the first time, an OpenCL based standalone read mapping tool, CORAL, is proposed. OpenCL computing framework assists usage of all the hardware resources available on a heterogeneous system for performance gains.

Genome is composed of four bases, viz. adenine, cytosine, thymine and guanine, represented as characters (`A C G T`) and require integer based operations for processing. For that purpose, simpler cores found on the embedded System-on-Chip (SoC) platforms may be better suited than complex general purpose CPU cores. MESGA [12] originally explored read mapping on MPSoC based embedded System. They implemented an existing mapping tool, BWA-aln [6], on 16 processors with 2 GB memory each and showed 7× speed-up compared to linear pipeline on an Intel server. However, their results were demonstrated on a cycle-accurate simulator rather than a real hardware platform. Besides, the results presented were only for 1 million simulated reads without any energy measurement. In this paper, we propose an OpenCL based REad maPper for heterogeneoUs sysTEms (REPUTE). REPUTE is a cross-platform tool capable of maximizing parallelization on multiple OpenCL conformant devices. REPUTE uses a memory optimized DP based filtration method inspired by the Optimum Seed Solver (OSS) method [13]. In comparison, CORAL, which is also based on an OpenCL framework, uses a heuristic based variable length *k-mer* selection criteria. DP based filtration in REPUTE improves specificity compared to heuristic approach as it examines the entire read to determine *k-mer* lengths to minimize the total number of candidate locations, while CORAL examines *k-*

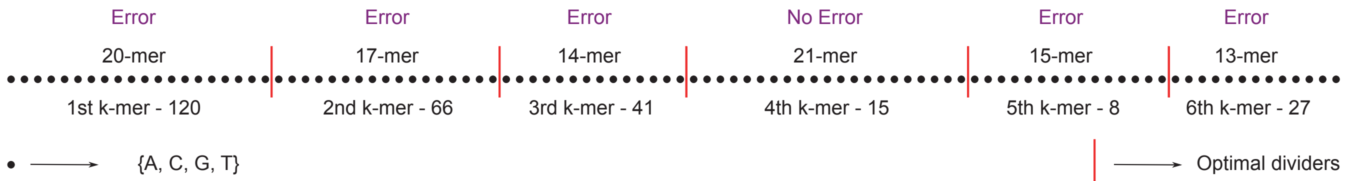


Fig. 1. Demonstration of pigeonhole principle for ($n = 100, \delta = 5$), where n is read length and δ is error. *K-mers* with their respective number of candidate locations are displayed. The vertical lines are the optimal dividers, identified in filtration stage, to minimize the total number of candidate locations. The dots represent one of the four bases: {`A, C, G, T`}.

mers serially. In addition, the REPUTE kernel flow has been modified to minimize the increase in memory footprint due to DP based filtration. Unlike the state-of-the-art mappers, our OpenCL based implementation enables us to demonstrate performance gains by launching parallel kernel executions on multiple devices. We have adopted a hardware-software co-design based approach to design a hardware-aware kernel with low memory footprint. We compare REPUTE with CORAL, GEM, Hobbes3, RazerS3, BWA-MEM and Yara [3]–[8] by mapping 2 million real reads to chromosome 21 on two separate systems: 1) Intel CPU + 2×Nvidia GPUs; 2) HiKey970 embedded SoC with ARM Cortex-A73/A53 cores. We demonstrate that REPUTE is up to 13× faster than existing mappers with similar accuracy on System 1 and consumes up to 27× less energy on embedded SoC, with comparable performance and similar accuracy. To the best of our knowledge, this is the first work which demonstrates the possibilities and potential of Embedded Genomics. The source code can be found at: https://github.com/nclaes/REPUTE

This rest of the paper is organized as follows. Section II presents DP based filtration method and an overview of the mapping process. Section III explains the experimental setup and the measurement methodology for performance, accuracy, power and energy. In Section IV, we present and discuss the results. Section V concludes the paper.

II. Methodology

Sequencing process fragments genome randomly into smaller sections and identifies them as small strings called reads. To re-assemble the genome, these reads are mapped to a reference genome with an aim to find the possible candidate locations from where it may have originated in the actual genome. Approximate string matching is used to account for errors and variations between the actual and the reference genome. There are three stages in read mapping: Preprocessing, Filtration and Verification. We discuss Preprocessing and Verification, briefly, as they employ standard methods used by most modern read mappers. We concentrate our efforts, mainly, on the DP based Filtration method, which is the bottleneck to performance gains.

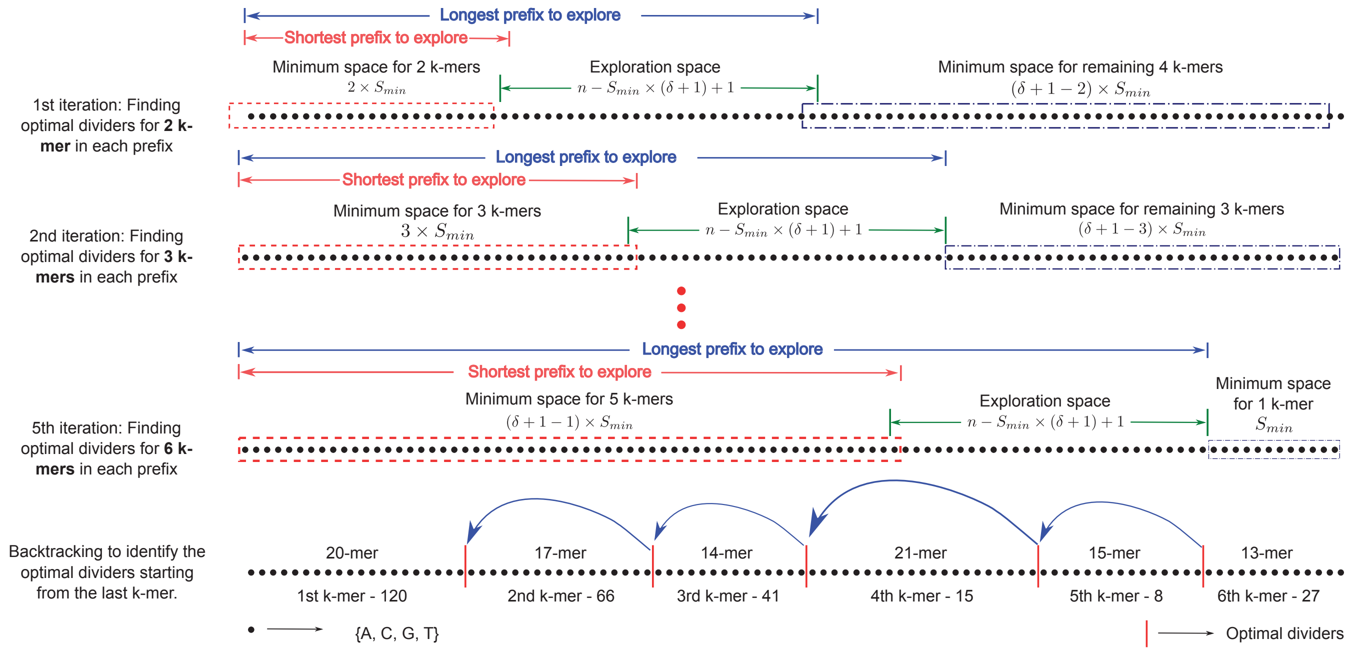
A. Preprocessing and Verification

The aim of the Preprocessing stage is to facilitate quick retrieval of information in the reference genome to speed-up the Filtration stage. We use FM-Index [14] and suffix array [15] based data structures to store the reference genome. These data structures have been, previously, used in many mappers including GEM, Yara, CORAL and BWA-MEM. To search candidate locations for reads in the reference genome, a k long section of the read, known as a *k-mer*, is searched using FM-Index backward search method. Upon a successful match, the location is obtained from the Suffix Array. This location points to a section in the reference genome where the entire read may align within the specified edit distance. Edit distance or error are synonymous in this paper. For further details interested readers may refer to [8], [14], [15]. The alignment is performed in the verification stage which employs a variant of the semi-global DP algorithm, known as Myer's bit vector algorithm, is used. It is one of the fastest and widely used method whose details can be found in [4], [16].

B. Dynamic Programming based Filtration

Pigeonhole principle [4] states that δ errors cannot occur in more than δ sections of the read. Therefore, dividing a read in $\delta + 1$ sections will leave a section error free, which should, ideally, match exactly in the reference genome at locations where it is suppose to have originated. Fig. 1 demonstrates pigeonhole principle for read length $n = 100$ and $\delta = 5$, divided into *k-mers* with different lengths (k). As the error-free *k-mer* is not pre-known, all the $\delta + 1$ *k-mers* are scanned in the reference genome and all candidate locations are verified. Each *k-mer* produces a different number of candidate locations depending on the starting, ending positions and their lengths. The vertical lines, in Fig. 1, are called optimal dividers. The optimal dividers partition the read into optimum length *k-mers* to minimize candidate locations, for better performance. Any filtration method proposed aims to find an optimum set of *k-mers*. RazerS3 and Hobbes3 use hashing based method to store and retrieve reference genome while CORAL, Yara, BWA-MEM and GEM use FM-Index based methods.

Fig. 2 demonstrates our memory optimized DP based filtration method. To begin with, the read parameters of (n, δ) and minimum *k-mer* length S_{min} are specified. The algorithm requires δ iterations for $\delta + 1$ *k-mers*. In each iteration, a fixed number of prefixes are explored, called the exploration space ($n - S_{min} \times (\delta + 1)$). The exploration space ensures that the minimum length of *k-mers* is not violated. Within an iteration, each prefix is divided into two sections: 1^{st} and 2^{nd}, with the objective of identifying optimal divider for the two sections, of each prefix, in the exploration space. The first iteration aims to find optimal divider for first two *k-mers*. Therefore, in this case, the 1^{st} section is the 1^{st} *k-mer* and 2^{nd} section is the 2^{nd} *k-mer*. The algorithm, then, finds optimal divider for each prefix starting from the longest to the shortest, as shown in



Longest prefix to explore
Shortest prefix to explore
Minimum space for 2 k-mers
$2 \times S_{min}$
Exploration space
$n - S_{min} \times (\delta + 1) + 1$
Minimum space for remaining 4 k-mers
$(\delta + 1 - 2) \times S_{min}$
1st iteration: Finding optimal dividers for 2 k-mer in each prefix
Minimum space for 3 k-mers
$3 \times S_{min}$
Minimum space for remaining 3 k-mers
$(\delta + 1 - 3) \times S_{min}$
2nd iteration: Finding optimal dividers for 3 k-mers in each prefix
Minimum space for 5 k-mers
$(\delta + 1 - 1) \times S_{min}$
Minimum space for 1 k-mer
S_{min}
5th iteration: Finding optimal dividers for 6 k-mers in each prefix
Backtracking to identify the optimal dividers starting from the last k-mer.
20-mer
17-mer
14-mer
21-mer
15-mer
13-mer
1st k-mer - 120
2nd k-mer - 66
3rd k-mer - 41
4th k-mer - 15
5th k-mer - 8
6th k-mer - 27
{A, C, G, T}
Optimal dividers

Fig. 2. Demonstration of memory optimised dynamic programming based filtration algorithm for parameters ($n = 100, \delta = 5$). δ iterations are required to obtain optimal dividers for $\delta + 1$ *k-mers*. In the end, the optimal dividers are found starting from the last one using backtracking.

Fig. 2. The second iteration carries forward the solution of first iteration to identify the optimal dividers for first three *k-mers*. However, the difference, here, is that the 1^{st} section, now, consists of first two *k-mers* combined while the 2^{nd} section is the 3^{rd} *k-mer*. Likewise, in the last iteration, the 1^{st} section will consist of first δ *k-mers* and the 2^{nd} section will be $(\delta+1)^{th}$ *k-mer*. At the end of all iterations, the backtracking process results in optimal divider for all the *k-mers*, as shown in Fig. 2.

For the DP based approach, the optimal dividers for all the prefixes in each iteration is required to be stored for backtracking in the end. This along with several other additional variables required by the algorithm, considerably, increases the memory footprint of the kernel. To avoid this, contrary to OSS, we have limited the exploration space from the entire read to the minimum space required. Among others, we optimized the bitwidths of variables to reduced memory footprint used FM-Index backward search in an efficient way to reduce memory accesses. However, it should be noted that the memory footprint varies with the size of the exploration space and, hence, depends on the n, δ and S_{min}. For smaller S_{min}, highly optimized solution can be obtained at the cost of greater memory footprint and filtration time, while a larger S_{min} results in lower footprint but, relatively, longer mapping time due to greater number of candidate locations. Inspired from OSS, we have retained all the optimizations proposed in [13]. To understand the approach in greater detail, interested readers can refer to [13].

III. Experimental Setup

The host program of REPUTE is written in `Python` and the kernel is in `C`. We use `PyOpenCL` because `Python` enables fast modifications and prototyping, yet, not affecting the mapping process. We compare REPUTE with CORAL, RazerS3 (`3.5.8`), Hobbes3 (`3.0`), Yara(`1.0.2`), BWA-MEM (`0.7.17`) and GEM (`3`). All mappers map 1 million (M) real reads each from NCBI databases: ERR012100_1 and SRR826460_1, to chromosome 21 of the Human Genome (version GRCh38/hg38) [17]. These databases consists of reads with lengths 100 and 150, respectively, and are mapped for error range (or edit distance) of 3-7. We use the following two platforms to map reads:

System 1: Intel Core i7-2600 CPU @ 3.40GHz, 16GB RAM and 2 $\times$ GeForce GTX 590, 1.5 GB RAM.
System 2: HiKey970 embedded SoC with ARM Cortex-A73 MPCore4 @up to 2.36GHz, ARM Cortex-A53 MPCore4 @up to 1.8GHz and 6 GB RAM.

We have used OpenCL 1.2 standard for portability across variety of platforms. This standard, however, imposes the following two restrictions:

a) It does not permit dynamic memory allocation. Hence, the number of outputs per read requires to be specified beforehand, in order to allocate sufficient memory.
b) The maximum amount of memory that can allocated to a variable is $(1/4)^{th}$ of the RAM capacity.

The aforementioned restrictions limit the maximum number of mapping locations per read due to the memory available to a variable. Thus, REPUTE reports the *first-n* mapping locations. It should be noted that we have compared, only, the mapping times wherever possible and all mappers used for comparison were configured to their recommended settings, unless specified. To ensure a comprehensive comparison, we have designed the following experiments.

A. Homogeneous Scenario

In this experiment, we run the mappers on the CPU of System 1. SAM file from RazerS3 is used as the gold standard,

TABLE I
THE RESULTS OF MAPPING 2M REAL READS TO CHROMOSOME (CHR) 21 ON THE CPU. T REPRESENTS MAPPING TIME IN SECONDS AND A REPRESENTS ACCURACY, MEASURED IN ACCORDANCE WITH SECTION III-A.

Chromosome 21 System 1	Read length	100						150					
	Error	3		4		5		5		6		7	
	Time/Accuracy	T(s)	A(%)	T(s)	A(%)	T(s)	A(%)	T(s)	A(%)	T(s)	A(%)	T(s)	A(%)
Intel Core i7-2600 CPU@3.40GHz, 16GB RAM + 2 × GeForce GTX 590, 1.5 GB RAM	RazerS3	26.7	100	42.6	100	65.7	100	30.7	100	50.6	100	91.3	100
	Hobber3	21.6	100	18.6	100	16.6	100	58.4	100	50	100	40.7	100
	Yara	10	5.22	21	4.51	25.5	4.00	38.2	5.27	116.5	4.54	321.4	4.14
	BWA-MEM	T(s) - 82.2			A(%) - 39.94			T(s) - 159.1			A(%) - 30.82		
	GEM	22	4.88	22	4.14	21	3.59	56	4.74	54	4.15	53	3.68
	CORAL-cpu	7.03	99.96	16.34	99.91	32.29	99.87	17.31	100	37.36	100	66.35	100
	REPUTE-cpu	**7.49**	**99.99**	**14.88**	**99.98**	**24.92**	**99.94**	**13.75**	**100**	**21.1**	**100**	**33.4**	**99.99**

TABLE II
THE RESULTS OF MAPPING 2M REAL READS TO CHROMOSOME (CHR) 21 ON THE CPU + GPU. T REPRESENTS MAPPING TIME IN SECONDS AND A REPRESENTS ACCURACY, MEASURED IN ACCORDANCE WITH SECTION III-B.

Chromosome 21 System 1	Read length	100						150					
	Error	3		4		5		5		6		7	
	Time/Accuracy	T(s)	A(%)	T(s)	A(%)	T(s)	A(%)	T(s)	A(%)	T(s)	A(%)	T(s)	A(%)
Intel Core i7-2600 CPU@3.40GHz, 16GB RAM + 2 × GeForce GTX 590, 1.5 GB RAM	RazerS3	26.7	100	42.6	100	65.7	100	30.7	100	50.6	100	91.3	100
	Hobber3	20.4	100	16.9	100	14.6	100	58.2	100	49.5	100	40.5	100
	Yara	10	99.2	21	99.4	25.5	99.5	38.2	100	116.5	100	321.4	100
	BWA-MEM	T(s) - 82.2			A(%) - 97.16			T(s) - 159.1			A(%) - 95.09		
	GEM	22	92.9	22	91.4	22	89.4	54	90.2	54	91.3	53	89.1
	CORAL-all	5.24	99.98	9.74	99.97	24.73	99.98	12.2	100	29.47	100	56.05	100
	REPUTE-all	**5.27**	**99.99**	**12.65**	**99.99**	**19.8**	**99.9**	**7.87**	**100**	**12.9**	**100**	**23.9**	**100**

as it is an *all-mapper* with high accuracy and has been used in Hobbes3 and Yara, previously. We configure RazerS3 to report a maximum of 100 mapping locations per read while other mappers produce up to 1000 locations per read. As the number of mapping locations reported by any mapper for any read cannot be pre-determined, hence, we set a limit on the maximum number of locations. Yara and BWA-MEM were configured to report all locations as they are *best-mappers* and can either produce the best mapping location or all the locations. To determine the mapping accuracy, all the mapping locations reported by the gold standard per read is searched in the output of other mappers. Along with the mapping locations the genome strand that the reads were mapped to are, also, matched.

B. Heterogeneous Scenario

In this experiment, we execute REPUTE on both CPU and GPU. Due to limited RAM of 1.5 GB on the GPUs and fairness of comparison, all mappers report 100 locations per read excluding Yara and BWA-MEM, which report all the mapping locations. Unlike state-of-the-art mappers, REPUTE distributes the workload on CPU and GPU, as per user specification, executing the work-items in task-parallel fashion using OpenCL framework. To obtain accuracy measurements, we employ a method similar to *any-best* scenario of the Rabema benchmark [18]. In contrast to Section III-A, we identify if all the reads mapped by the gold standard have been reported by other mappers with at least one matching mapping location and strand.

C. Embedded Scenario

HiKey970 SoC boots with a Linux distribution, `Lebuntu`, provided by the manufacturer. It needs to be flash booted instead of a bootable SD card. With limited onboard flash memory, we could install only a limited number updates and libraries. Among other mapping tools, we could successfully run, only, RazerS3, Hobbes3, CORAL and REPUTE on HiKey970. We adopt the same measurement methodology as stated in Section III-B.

D. Power and Energy Consumption

We compare Hobbes3, RazerS3, CORAL and REPUTE for energy efficiency using a power meter at the power source of the two systems. We measure the average power consumption during the mapping process and subtract it with the idle power to measure the power consumption during mapping process. We multiply the power consumption with mapping time to measure energy consumption. For a fair comparison between homogeneous and heterogeneous scenarios, we need to distribute, approximately, equal amounts of reads between the CPU and GPUs. For this purpose, we chose following cases to take the measurements: $n = 100, \delta = 3$ and $n = 150, \delta = 5$. The former case maps 480,000 reads and the latter maps 500,000 reads on the GPU using REPUTE.

TABLE III
READ MAPPING ON THE HIKEY970 SOC. T - TIME IN SECONDS AND A - ACCURACY, MEASURED IN ACCORDANCE WITH SECTION III-C

Chromosome 21 System 2	Read length	100						150					
	Error	3		4		5		5		6		7	
	Time/Accuracy	T(s)	A(%)	T(s)	A(%)	T(s)	A(%)	T(s)	A(%)	T(s)	A(%)	T(s)	A(%)
ARM Cortex-A73, A53, 8 cores, 6 GB RAM	RazerS3	89.1	100	127.5	100	222.3	100	96.8	100	168.1	100	328.1	100
	Hobber3	54.06	100	47.37	100	46.68	100	89.95	100	78.21	100	69.34	100
	CORAL-HiKey	16.41	100	38.39	100	67.48	100	38.65	100	78.50	100	134.1	100
	REPUTE-Hikey	**17.47**	**99.99**	**35.35**	**99.99**	**60.61**	**99.99**	**49.44**	**100**	**56.3**	**100**	**84.72**	**100**

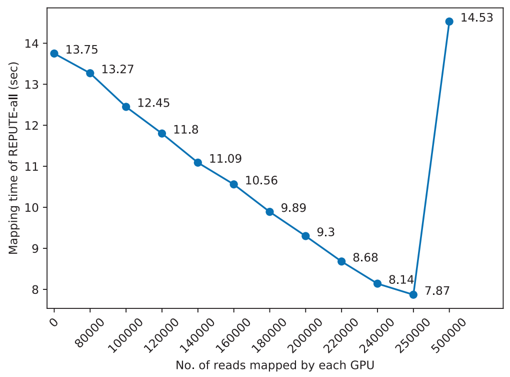


Fig. 3. Mapping time for different distributions of workloads on CPU and GPU for ($n = 150, \delta = 5$) and minimum *k-mer* length of 22. X-axis shows the number of reads, out of 1 million, mapped by each GPU and the remaining reads are mapped on the CPU.

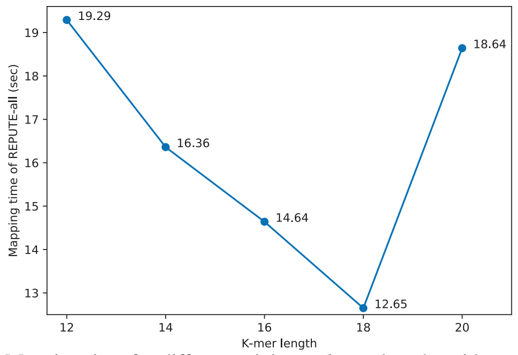


Fig. 4. Mapping time for different minimum *k-mer* lengths with same distributions of workloads on CPU and GPU. CPU mapped 820,000 reads and GPU mapped 90,000 reads each with the read configuration of ($n = 100, \delta = 4$).

IV. RESULTS AND DISCUSSION

Table I presents the results of the Homogeneous scenario mentioned in Section III-A. It is evident that REPUTE outperforms RazerS3, Yara, BWA-MEM on both runtimes and accuracy for all error profiles. REPUTE is up to 13× faster than Yara. RazerS3 is configured to produce 100 outputs per read compared to 1000 outputs for other mappers, hence, reducing its mapping time significantly. Except for ($n = 100$, $\delta = 5$), REPUTE performs better than Hobbes3 and GEM. The mapping accuracy of REPUTE is considerably better than GEM and is equal or comparable to Hobbes3. We can see that REPUTE provides upto 4× speedup over Hobbes3 for lower error profiles and longer read lengths. Compared to CORAL, DP based filtration has reduced the mapping time, especially, for longer read lengths and high error profiles.

Table II presents the results of the Heterogeneous scenario discussed in Section III-B. The performance of REPUTE compared to other mappers follow similar trends as discussed in the previous paragraph. The contrast between the mapping times in REPUTE-cpu and REPUTE-all demonstrate that performance can be enhanced by using multiple devices in parallel. Using GPUs, we obtained an additional speedup of up to ≈2× making REPUTE up to 7× faster than Hobbes3 for longer reads and smaller error profiles.

REPUTE-all uses CPU along with two Nvidia GPUs to distribute the workload and map reads in task parallel fashion. It launches the kernels simultaneously and upon completion it combines the results, thus, making one of the devices the performance bottleneck. The distribution of workload among various devices, hence, should be performed judiciously to obtain optimum performance. The maximum number of reads that can be mapped on a device is limited by the private memory available to the compute cores and the global memory or RAM available to the device. Fig. 3 presents a scenario of performance gains by offloading more workloads to GPU for given ($n = 150, \delta = 5$) and fixed minimum *k-mer* length of 22. Large *k-mer* lengths reduces the memory footprint of the kernel allowing more workgroups to be processed by the GPU without running out of resources. The extreme point on the left in Fig. 3, indicates the mapping time when GPU is not used while the rightmost point gives the mapping time when all reads are mapped on the GPUs. We can see that utilizing GPUs, which are commonly found on most modern computing systems, can provide additional performance gains.

Fig. 4 shows the mapping times for different minimum *k-mer* lengths with a constant workload distribution between CPU (820,000 reads) and GPU (90,000). For smaller *k-mer* lengths, the mapping time is higher due to exploration of larger number of possibilities in the DP based filtration method. As the *k-mer* lengths increase, the time taken in filtration decreases, still producing similar number of candidate locations per read. However, for larger *k-mer* length of 20, the exploration space to select the optimum divisions of read is fewer, thus, resulting in higher candidate locations per read and, therefore, longer mapping time. The results presented in Table I and II are the best performances of REPUTE taking into consideration the *k-mer* lengths and workload distribution.

Table III presents the results for the embedded Scenario mentioned in Section III-C. We can see that for ($n = 100, \delta = 5$) and ($n = 150, \delta = 7$), REPUTE performs comparable to Hobbes3 with similar accuracy while for other cases it

TABLE IV
ENERGY CONSUMPTION IN ACCORDANCE WITH SECTION III-D.

	$n = 100, \delta = 3$		$n = 150, \delta = 5$	
	P(W)	E(J)	P(W)	E(J)
	System 1 - 160 W (Idle power)			
RazerS3	241	2162.7	243	2548.1
Hobbes3	254	1917.6	258	5703.6
CORAL-CPU	365	1440.1	371	3652.3
CORAL-all	454	1540.7	461	3673.1
REPUTE-CPU	354	1691.5	358	2859.1
REPUTE-all	455	1554.7	490	2597.1
	System 2 - 3.5 W (Idle power)			
RazerS3	7.5	356.3	8.6	493.5
Hobbes3	7.5	216.2	8.4	440.8
CORAL-HiKey	8.5	82.06	9.1	216.5
REPUTE-HiKey	8	78.6	7.8	212.6

outperforms Hobbes3. REPUTE is up to 4× times faster than RazerS3. Table IV presents the power and energy measurements mentioned in Section III-D. We observe that REPUTE-all, which distributes workload on CPU and GPU, uses more power but less energy and is faster than other mappers including REPUTE-cpu. On Hikey970, REPUTE outperforms other mappers, significantly, on the energy consumption. However, the most important takeaway, in our opinion, is that exploration of genomic computations on embedded SoCs can produce significant energy savings which can lower down the overall cost of whole genome sequencing. We demonstrate energy savings of upto 20× on HiKey970 embedded SoC compared to general purpose workstations, as shown in Table IV.

Currently, REPUTE is tailored to map short reads of length 100-150, even though the algorithm does not impose any such restrictions *per se*. REPUTE gives the mapping positions, edit distance and strand for each but, currently, does not produce the CIGAR string. The memory footprint of REPUTE is large due to limitations on dynamic memory allocation posed by the OpenCL standard. Hence, depending on the RAM available, we have to limit the number of mappings per read or run the kernel multiple times with smaller read sets. Another reason for large memory footprint is the size of the FM-Index data structure and suffix array. This, however, can be significantly reduced by storing elements after fixed intervals as used in [19]. We envisage that the future versions of REPUTE will deliver significantly reduce memory footprint and SAM output format.

V. CONCLUSIONS

We proposed a cross-platform OpenCL based REad maPper for heterogeneoUs sysTEms (REPUTE) capable of parallel kernel executions on multiple devices. REPUTE uses a memory optimized dynamic programming based algorithm for performance driven pruning of reference genome to map short reads. We have compared REPUTE with state-of-the-art read mappers using real human reads on two different platforms. REPUTE outperforms other mappers on performance and accuracy parameters in most of the cases. We have demonstrated, for the first time, the potential of Embedded genomics for energy efficiency without loss of accuracy and comparable performance. Compared to other mappers, REPUTE provides better significant energy savings. Our results show that moving genomics from high-performance servers and workstations to embedded systems can potentially unleash new opportunities for low-cost genomics.

REFERENCES

[1] L. Hood and D. Galas, "P4 medicine: Personalized, predictive, preventive, participatory a change of view that changes everything," *Computing community consortium*, 2008.

[2] K. Reinert, B. Langmead, D. Weese, and D. J. Evers, "Alignment of next-generation sequencing reads," *Annual Review of Genomics and Human Genetics*, vol. 16, no. 1, pp. 133–151, 2015, pMID: 25939052.

[3] S. Marco-Sola, M. Sammeth, R. Guigo, and P. Ribeca, "The gem mapper: fast, accurate and versatile alignment by filtration," *Nature Methods*, vol. 9, pp. 1185–1188, 2012.

[4] D. Weese, M. Holtgrewe, and K. Reinert, "Razers 3: Faster, fully sensitive read mapping," *Bioinformatics*, vol. 28, no. 20, pp. 2592–2599, 2012.

[5] J. Kim, C. Li, and X. Xie, "Hobbes3: Dynamic generation of variable-length signatures for efficient approximate subsequence mappings," pp. 169–180, 2016.

[6] H. Li and R. Durbin, "Fast and accurate long-read alignment with burrowswheeler transform," *Bioinformatics*, vol. 26, no. 5, pp. 589–595, 2010. [Online]. Available: http://dx.doi.org/10.1093/bioinformatics/btp698

[7] E. Siragusa, "Approximate string matching for high-throughput sequencing," *Free University of Berlin*, 2015.

[8] 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 Transactions on Computational Biology and Bioinformatics*, pp. 1–1, 2019.

[9] Z. D. Stephens, S. Y. Lee, F. Faghri, R. H. Campbell, C. Zhai, M. J. Efron, R. Iyer, M. C. Schatz, S. Sinha, and G. E. Robinson, "Big data: Astronomical or genomical?" *PLOS Biology*, vol. 13, no. 7, pp. 1–11, 07 2015.

[10] W. V. Heddeghem, S. Lambert, B. Lannoo, D. Colle, M. Pickavet, and P. Demeester, "Trends in worldwide ict electricity consumption from 2007 to 2012," *Computer Communications*, vol. 50, pp. 64 – 76, 2014, green Networking. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140366414000619

[11] S. Aluru and N. Jammula, "A review of hardware acceleration for computational genomics," *IEEE Design Test*, vol. 31, no. 1, pp. 19–30, Feb 2014.

[12] V. Gnanasambandapillai, A. Bayat, and S. Parameswaran, "Mesga: An mpsoc based embedded system solution for short read genome alignment," in *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan 2018, pp. 52–57.

[13] H. Xin, S. Nahar, R. Zhu, J. Emmons, G. Pekhimenko, C. Kingsford, C. Alkan, and O. Mutlu, "Optimal seed solver: optimizing seed selection in read mapping," *Bioinformatics*, vol. 32, no. 11, pp. 1632–1642, 2016.

[14] P. Ferragina and G. Manzini, "Opportunistic data structures with applications," in *Proceedings 41st Annual Symposium on Foundations of Computer Science*, 2000, pp. 390–398.

[15] U. Manber and G. Myers, "Suffix arrays: A new method for on-line string searches," *SIAM Journal on Computing*, vol. 22, no. 5, pp. 935–948, 1993. [Online]. Available: http://dx.doi.org/10.1137/0222058

[16] G. Myers, "A fast bit-vector algorithm for approximate string matching based on dynamic programming," *J. ACM*, vol. 46, no. 3, pp. 395–415, May 1999.

[17] W. J. Kent, C. W. Sugnet, T. S. Furey, K. M. Roskin, T. H. Pringle, A. M. Zahler, Haussler, and David, "The human genome browser at ucsc," *Genome Research*, vol. 12, no. 6, pp. 996–1006, 2002.

[18] 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, no. 1, p. 210, May 2011.

[19] B. Langmead and S. L. Salzberg, "Fast gapped-read alignment with bowtie 2," *Nature methods*, vol. 9, no. 4, pp. 357–359, 2012.