# PLEDGER: Embedded Whole Genome Read Mapping using Algorithm-HW Co-design and Memory-aware Implementation

Sidharth Maheshwari†, Rishad Shafik†, Ian Wilson†, Alex Yakovlev†, Venkateshwarlu Y. Gudur‡ and Amit Acharyya◇
†Newcastle University, Newcastle upon Tyne, UK.
◇Department of Electrical Engineering, Indian Institute of Technology Hyderabad, Hyderabad, India.
E-mail: S.Maheshwari2@newcastle.ac.uk, Rishad.Shafik@newcastle.ac.uk

2021 Design, Automation & Test in Europe Conference & Exhibition (DATE) | 978-3-9819263-5-4/21/$31.00 ©2021110.23919/DATE51398.2021.9473909

***Abstract*—With over 6000 known genetic disorders, genomics is a key driver to transform the current generation of healthcare from reactive to personalized, predictive, preventive and participatory (P4) form. High throughput sequencing technologies produce large volumes of genomic data, making genome reassembly and analysis computationally expensive in terms of performance and energy. In this paper, we propose an algorithm-hardware co-design driven acceleration approach for enabling translational genomics. Core to our approach is a Pyopencl based tooL for gEnomic workloaDs tarGeting Embedded platforms (PLEDGER). PLEDGER is a scalable, portable and energy-efficient solution to genomics targeting low-cost embedded platforms. It is a read mapping tool to reassemble genome, which is a crucial prerequisite to genomics. Using bit-vectors and variable level optimisations, we propose a low-memory footprint, dynamic programming based filtration and verification kernel capable of accelerated parallel heterogeneous executions. We demonstrate, for the first time, mapping of real reads to whole human genome on a memory-restricted embedded platform using novel memory-aware preprocessed data structures. We compare the performance and accuracy of PLEDGER with state-of-the-art RazerS3, Hobbes3, CORAL and REPUTE on two systems: 1) Intel i7-8750H CPU + Nvidia GTX 1050 Ti; 2) Odroid N2 with 6 cores: 4×Cortex-A73 + 2×Cortex-A53 and Mali GPU. PLEDGER demonstrates persistent energy and accuracy advantages compared to state-of-the-art read mappers producing up to 11× speedups and 5.9× energy savings compared to state-of-the-art hardware resources.**

***Index Terms*—OpenCL, embedded genomics, read mapping, heterogeneous computing, low-memory footprint, energy efficient.**

## I. Introduction

Genomics is a crucial application in medicine with the potential to open up significant opportunities in prognosis and therapy of over 6000 single-gene genetic conditions and other diseases. It is a major driver for the next generation of personalized, predictive, preventive and participatory (P4) medicine [1]. Prerequisite to genomics is the availability of genome which is obtained from the sequencing and assembly pipelines of the whole genome sequencing (WGS) [2]. Sequencing process produces fixed-length small subsections of genome, called reads, which are then reassembled to obtain the original genome. With the advent of high throughput sequencing (HTS) technologies, the cost of sequencing has significantly reduced, however, at the expense of processing massive amounts of data [3]. Though, HTS has revolutionized WGS, it has contributed to making it a Big Data application with computational constraints [3]. The continuous growth in data has strained the available computational resources, thereby, consumig large amounts of energy [4]. Indeed, the demand of scaling up the computational capabilities energy efficiently is hindering the progress of this crucial and emerging new application [5].

To reassemble genome, a performance-driven read-alignment approach [6] is used which maps reads to an existing reference genome (RG) using read mapping tools. The mapping process engages approximate string matching and dynamic programming (DP) algorithms in tandem with the RG, stored in the form of data structures following a tool-specific preprocessing strategy. Conventionally, most of the state-of-the-art read mappers, such as [7], [8], have been optimized for CPU and are oblivious to other hardware resources available in modern heterogeneous systems such as the GPU. Several FPGA and GPU specific tools have been proposed, however, they are not flexible to changes in parameters and, often, require platform-specific programming skills [9].

CORAL [10] demonstrates an OpenCL based heterogeneous read mapping scenario where workloads are distributed on available CPU+GPU for acceleration. It, however, uses a heuristic filtration methodology. REPUTE [5] proposes a DP based filtration methodology using OpenCL to improve performance and demonstrates an embedded implementation, for chr21, on HiKey970 platform with energy savings of 27× compared to a workstation. Both REPUTE and CORAL, however, use data structures with large memory footprints making it unfit to be used with longer chromosomes (e.g. chr1, chr2) in memory-restricted embedded platforms. Hobbes3 [8] uses a DP based filtration methodology along with heuristic schemes to optimize performance on q-gram inverted index for high-performance. RazerS3 [7] is accuracy focused, commonly used as gold standard for comparison but it does not employ any data structures to accelerate read mapping. As such, it is slower than other read mappers.

In this paper, we propose a Pyopencl based tooL for gEnomic workloaDs tarGeting Embedded platforms (PLEDGER) to enable translational genomics. It employs a novel preprocessing scheme to generate memory-aware data structures on platforms with available RAM capacity of 3.6 GB. PLEDGER uses DP based filtration and verification kernel akin to REPUTE with improvements to use memory-aware data struc-

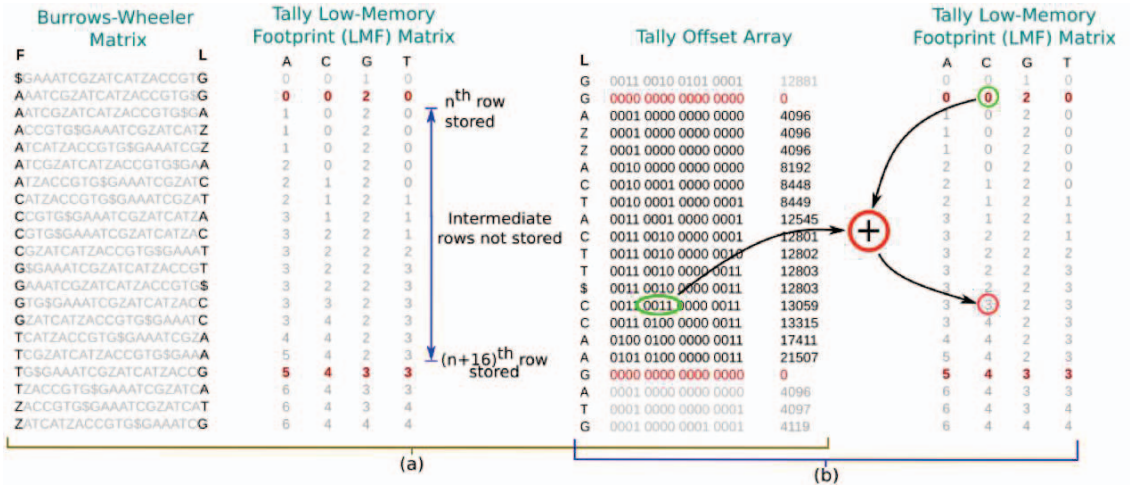


Fig. 1. (a) Demonstration of proposed low memory footprint (LMF) tally matrix along with the tally offset array. (b) Visualization of bit-vector operation to obtain the desired element of the tally matrix using tally LMF matrix and tally offset array during filtration in $O(1)$ time.

tures, which affects the performance as a trade-off. To improve performance, we use bit-vector operations and localized variable optimizations to minimize the memory footprint of the kernel. In CORAL and REPUTE, the mapping process needs to be repeated for each chromosome while PLEDGER is capable of mapping to all or user selected chromosomes: 1-22, X and Y, automatically, enhancing its portability. It is a stand-alone embedded tool capable of mapping entire genome on any CPU or GPU with over 3.6 GB available RAM. It, also, opens up new opportunities for embedded cluster acceleration for genomic workloads.

We compare PLEDGER with RazerS3, Hobbes3, CORAL and REPUTE by mapping 1 million (M) real human reads of lengths 100 and 150 each, to chromosomes 1-22, X and Y. We execute read mappers on two systems 1) Intel i7-8750H CPU, 16GB RAM + Nvidia GTX 1050 Ti, 4GB RAM; 2) Odroid N2 with quad-core ARM Cortex-A73 + dual core Cortex-A53, 4GB RAM. Among state-of-the-art read mappers, only Hobbes3 and RazerS3, although oriented towards CPU, ran successfully on HiKey970 (6GB RAM) for comparison in REPUTE [5]. However, with just 4GB RAM on Odroid N2, only Hobbes3 and PLEDGER could be benchmarked. We demonstrate up to 11× speedup with similar accuracy. Our embedded implementation consumes 5.9× less energy than state-of-the-art computing resource. The PLEDGER source code can be found at: https://github.com/nclaes/pledger.

## II. Methodology

The read mapping process consists of three stages: Preprocessing, Filtration and Verification. RG is preprocessed and stored as data structures suitable to the filtration scheme. It assists with rapid pruning of RG while searching for possible candidate locations for a read. These candidate locations are then verified(aligned) against the RG to find it's position of origination in the original genome during sequencing, if it exists. Verification is performed for $\delta$ error or edit distance, using a widely used variant of the semi-global DP algorithm, the Myer's bit vector algorithm [7], [11]. In the following subsections, we focus on the proposed preprocessing and filtration methodologies.

### A. Memory-aware preprocessing

We store the RG as FM-Index [12] and suffix array [13] based data structures. FM-Index backward search offers $O(n)$ time complexity to search a string of length $n$, making it one of the fastest approximate string matching algorithm. These data structures have been, previously, used in many mappers including BWA-MEM, CORAL and REPUTE. Fig. 1(a) visualizes the construction of FM-Index data structure for the string `GAAATCGZATCATZACCGTG$`. It involves the formation of tally matrix (TM) using **F** and **L** arrays obtained by applying the Burrows-Wheeler transform, refer [10] for more details. The length of TM is same as that of the genome and is the major bottleneck to low-memory implementations. For example, TM for chr1 requires about 4GB memory.

To reduce the size of TM, we can store a limited number of rows at fixed intervals. However, this necessitates availability of **L** array, which will be scanned during run-time, to reconstruct the missing rows by counting the number of occurrences of bases: `A C G T`. This count will be added to the previous available row to obtain the values of the desired row. The DP based filtration methodology used in this paper would require repetitive looping ($O(n)$) over the **L** array which will significantly increase the filtration time. We eliminate looping by encoding the **L** array as bit-vectors in an additional tally offset array. It is followed by a single bit-vector operation ($O(1)$) to obtain the value of the desired row in TM, as shown in Fig. 1(b). We store every $16^{th}$ row in the TM and store the number of occurrences of `A C G T` for the, corresponding, missing 15 rows in the **L** array using 4-bits each, resulting in a 16-bit unsigned integer. As preprocessing is one-off task, it does not affect the run-time and prevents looping. The proposed preprocessing scheme reduces the size of tally matrix by $\approx 5.5\times$, bringing 4 GB down to 746.9 MB for chr1.

TABLE I
MAPPING TIME(IN SECONDS) FOR 1M REAL READS,, OF LENGTH $n = 150$ AND ERROR $\delta = 7$,, TO CHROMOSOME (CHR) 1-22, X AND Y ON SYSTEM - 1 (CPU+GPU). PLEDGER-ALL DISTRIBUTES WORKLOAD OVER CPU AND GPU IN 4:1 RATIO WHILE OTHERS EXECUTE ONLY ON THE CPU.

| | chr1 | chr2 | chr3 | chr4 | chr5 | chr6 | chr7 | chr8 | chr9 | chr10 | chr11 | chr12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RazerS3 | 487 | 434.2 | 359.5 | 321 | 318.5 | 319 | 253 | 237.9 | 259.3 | 244.8 | 272.4 | 162.6 |
| Hobber3 | 40.6 | 34.4 | 32.4 | 30.8 | 31.3 | 31.4 | 32.4 | 30.6 | 30.5 | 31.1 | 31.4 | 31.7 |
| CORAL | 123.8 | 115.1 | 106.8 | 96.7 | 93 | 100.9 | 113.8 | 100.3 | 100.7 | 104.4 | 93.7 | 100 |
| REPUTE | 79 | 70.8 | 62.9 | 56.5 | 62.6 | 58.8 | 65.9 | 57.2 | 56.2 | 58.3 | 55.6 | 60.2 |
| PLEDGER-cpu | 72.4 | 71.4 | 66.9 | 61.8 | 63.4 | 63.8 | 65.6 | 58.8 | 58.5 | 60.5 | 58.7 | 61.8 |
| **PLEDGER-all** | **61.5** | **56.4** | **51.9** | **47.9** | **49.1** | **48.9** | **50.7** | **45.3** | **44.8** | **46.5** | **45.7** | **48.1** |

| | chr13 | chr14 | chr15 | chr16 | chr17 | chr18 | chr19 | chr20 | chr21 | chr22 | chrX | chrY |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RazerS3 | 162.6 | 175.4 | 177.8 | 175.9 | 231.5 | 129 | 210.4 | 137.9 | 71.8 | 105.7 | 280.9 | 43.5 |
| Hobber3 | 28.5 | 29.4 | 29.6 | 29.8 | 31 | 28 | 30.1 | 28.1 | 26.4 | 27.4 | 31.4 | 25.5 |
| CORAL | 75.4 | 89.8 | 91 | 98.6 | 105.3 | 74.4 | 102.6 | 80.4 | 51.3 | 75.7 | 102.6 | 31.7 |
| REPUTE | 40.3 | 48.1 | 48.8 | 53 | 60.3 | 39.6 | 56.8 | 41.9 | 29.9 | 75.7 | 102.6 | 31.7 |
| PLEDGER-cpu | 48.4 | 51.8 | 52.4 | 54.1 | 58 | 41.4 | 55.1 | 44.9 | 34.9 | 41.7 | 57.4 | 28.6 |
| **PLEDGER-all** | **37.7** | **39.6** | **40.7** | **43.3** | **46.3** | **34.6** | **44.8** | **36.1** | **28.3** | **33.6** | **46.3** | **23.6** |

### *B. Filtration for memory-aware data structures*

Read alignment approach widely uses the pigeonhole principle [7] which states that if a read, with $\delta$ error, is divided into non-overlapping $\delta+1$ sections, then one section would be left error-free and match exactly at its place of origination. Each non-overlapping section of length *k* are called *k-mers*, which are pruned through the RG using backward search to find the candidate locations. Filtration aims to identify suitable $\delta+1$ *k-mers* in a read to minimize the total candidate locations leading to reduced mapping time as there are fewer DP-based verification cycles. We improve over DP-based filtration method used in [5]; to use proposed memory-aware data structures with fixed bit-vector operations to obtain the desired value of TM as shown in Fig. 1(b). Both filtration and verification are done *in situ* in the same work-item (or thread).

## III. EXPERIMENTAL SETUP

The host code of PLEDGER is written in `Python` and the kernel in `C`. `Python` enables easy handling of strings and fast prototyping. We compare PLEDGER with CORAL, REPUTE, RazerS3 and Hobbes3. We use RazerS3 as the gold standard and use a method similar to *any-best* scenario of the Rabema benchmark [14] for accuracy comparison. For the same read, each reported location by the mapper is compared to those reported by RazerS3 and if any location and strand matches to the gold standard, we report it as an accurate match. All mappers map 1M real reads each from NCBI databases: ERR012100_1 and SRR826460_1, with read lengths $n = 100$ and 150, respectively, to chromosome 1-22, X and Y (GRCh38/hg38) [15]. We map reads with 5% error rate ($\delta$) on the following two platforms:

**System 1**: Intel i7-8750H CPU, 16GB RAM + Nvidia GTX 1050 Ti, 4GB RAM.

**System 2**: Odroid N2 with 4×Cortex-A73 + 2×Cortex-A53 and Mali-G52 GPU, 4GB RAM.

We use OpenCL 1.2 standard for cross-platform portability. OpenCL 1.2, however, does not allow dynamic memory allocation and the maximum memory that can be allocated to one variable cannot exceed $(1/4)^{th}$ of the RAM capacity. This had earlier restricted floating of large tally matrices in CORAL and REPUTE on memory restricted platforms. Similar to CORAL and REPUTE, PLEDGER reports the *first-n* mapping locations. We compare the mapping times of different mappers with their recommended settings and all mappers have been configured to report 100 mapping locations per read.

## IV. RESULTS AND DISCUSSION

In all our experiments, we have found that all mappers produced over 99% accuracy in reporting locations in comparison to the gold standard, as mentioned in Section III.

### *A. System 1 - CPU+GPU*

We conduct two experiments on System 1, where we compare PLEDGER with RazerS3, Hobbes3, CORAL and REPUTE: first, using only the CPU and, second, using both CPU and GPU. The memory requirements prohibit CORAL and REPUTE from running on GPU, however, PLEDGER can map all chromosomes on both CPU and GPU. We obtained results for both $n = 100$ and $n = 150$, both showing similar trends, but for the sake of brevity we present results, only, for $n = 150$ and $\delta = 7$ as shown in Table I. **PLEDGER-cpu** uses just the CPU while **PLEDGER-all** uses CPU+GPU by distributing workload in the ratio 4:1. The ratio was chosen upon empirical observation, heuristically, to give the best mapping times. We can see that PLEDGER-cpu and PLEDGER-all outperforms RazerS3 and CORAL for all chromosomes, producing 1.6-11× speedups. CORAL is slower due to its heuristic filtration methodology. PLEDGER performance is comparable to REPUTE while PLEDGER-all outperforms it by offloading the workload to GPU. Hobbes3 has outperformed PLEDGER in performance. The performance gap, however, narrows when PLEDGER distributes workload on available GPU using its parallel heterogeneous execution capabilities. For chromosome Y in Table I, we see that PLEDGER-all outperforms Hobbes3. Chromosome Y is the smallest of all chromosomes and we observe that PLEDGER's performance improves for smaller chromosomes.

### *B. System 2 - Odroid N2*

Among existing read mappers, only Hobbes3 and PLEDGER could run on a memory restricted Odroid N2. From

TABLE II
MAPPING TIME(IN SECONDS) FOR 1M REAL READS, OF LENGTH $n = 100$ AND ERROR $\delta = 5$, TO CHROMOSOME (CHR) 1-22, X AND Y ON SYSTEM - 2, ODROID N2 PLATFORM.

| | chr1 | chr2 | chr3 | chr4 | chr5 | chr6 | chr7 | chr8 | chr9 | chr10 | chr11 | chr12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Hobber3 | 88.5 | 82.1 | 73.63 | 69.8 | 75.5 | 62.8 | 65.1 | 57.5 | 56.5 | 59.3 | 56.8 | 59.8 |
| PLEDGER | 168.8 | 162.8 | 151.2 | 148.2 | 146.7 | 144 | 147.4 | 137.4 | 134.7 | 140.7 | 135.7 | 138.7 |

| | chr13 | chr14 | chr15 | chr16 | chr17 | chr18 | chr19 | chr20 | chr21 | chr22 | chrX | chrY |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Hobber3 | 48.1 | 63.6 | 50 | 52 | 58 | 48.4 | 50.9 | 45.9 | 37.3 | 40.8 | 59.8 | 35.4 |
| PLEDGER | 119.6 | 123.2 | 122.7 | 125.9 | 128.5 | 111.1 | 119.8 | 113.8 | 91.8 | 99.7 | 140.4 | 79.2 |

Table II, we can see that Hobbes3 outperforms PLEDGER in all cases. Even though PLEDGER is capable of using the Mali GPU but we did not find any additional performance gains. This is because of the on-board architecture, shared RAM and kernel design. Mali has low operational frequency (950 MHz) compared to Nvidia GPU (1392 MHz). Although, PLEDGER is slower compared to Hobbes3 it provides portability to OpenCL conformant devices and scalability for implementation on an embedded cluster to accelerate genomic workloads.

### C. Power and energy consumption

We compare the power and energy utilization of Hobbes3 and PLEDGER on system 1 and 2. We measure the average power consumption and deduct the idle power to measure the run-time power consumption. To measure energy consumption, we multiply the run-time power with the total mapping time for 24 chromosomes. From Table III, we observe that using embedded platform can lead to 4.34-5.93× energy savings compared to general purpose computers. It is, also, evident that high performance can directly yield huge energy savings in the embedded scenario.

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

| | $n = 100, \delta = 5$ | | $n = 150, \delta = 7$ | |
|---|---|---|---|---|
| | P(W) | E(J) | P(W) | E(J) |
| | System 1 - 20 W (Idle power) | | | |
| Hobbes3 | 79 | 20006.9 | 80 | 44028 |
| PLEDGER-cpu | 78 | 41035 | 79 | 78605.7 |
| PLEDGER-all | 113 | 51205.8 | 114 | 98859.8 |
| | System 2 - 3 W (Idle power) | | | |
| Hobbes3 | 6.6 | 4611.8 | 6.6 | 7422.69 |
| PLEDGER | 6 | 9396 | 6.1 | 18404.7 |

## V. CONCLUSIONS AND FUTURE WORK

We present a Pyopencl based tooL for gEnomic workloaDs tarGeting Embedded platforms (PLEDGER). It is a standalone tool capable of completing entire mapping process to the whole human genome in a memory-restricted ($\geq$ 3.6 GB) embedded environment. It's portable, scalable and enables parallel heterogeneous executions on both CPU and GPU. It uses memory-aware data structures and algorithm-hardware co-design to target embedded scenarios for energy efficiency. It uses bit-vector operations and memory optimized dynamic programming based algorithm to accelerate the mapping process. We compare PLEDGER with state-of-the-art read mappers and demonstrate significant performance gains and energy savings. PLEDGER, however, has scope of significant improvements in performance as it verifies 3-5× more locations per read compared to state-of-the-art Hobbes3. In our future work, we intend to append our filtration scheme with post-filtration optimizations to increase the specificity of selection of candidate locations.

## 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] H. Ye, J. Meehan, W. Tong, and H. Hong, "Alignment of short reads: A crucial step for application of next-generation sequencing data in precision medicine," *Pharmaceutics*, vol. 7, no. 4, pp. 523–541, 2015.

[3] 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.

[4] 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

[5] S. Maheshwari, R. Shafik, I. Wilson, A. Yakovlev, and A. Acharyya, "Repute: An opencl based read mapping tool for embedded genomics," in *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2020, pp. 121–126.

[6] 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.

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

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

[9] 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.

[10] 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.

[11] 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.

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

[13] 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

[14] 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.

[15] 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.