Understanding and Characterization of Pangenomics

Noah Kaplan*, Yufeng Gu, Reetuparna Das
Dept. of Computer Science and Engineering
University of Michigan – Ann Arbor
Michigan, United States
*Corresponding Author: kaplannp@umich.edu

***Abstract*—Genome sequencing is a key component of precision health, allowing us to tailor treatment to individuals based on their genetics. Reference-guided assembly is the most common for complete human genome construction, but most genomes are sequenced using a linear reference, which does not encapsulate the diversity of our species. This leads to poor alignment quality and variant detection.**

Alignment to a graph composed of multiple references, *pangenomics*, presents a promising alternative, but introduces a new set of tools with different, and often greater computational requirements. These workloads, and the increasing volume of sequencing data, require specialized hardware to keep up with demand. To design and evaluate such hardware, insights are needed into the characterization and bottlenecks of common pangenomics workloads. In this work, we present an overview of the pangenomics pipeline and an algorithmic overview of two leading pangenomics alignment tools for short and long reads respectively. We also provide a timing and microarchitecture analysis of these tools to inform future research.

***Index Terms*—pangenomics, benchmarking, alignment**

I. Introduction

Genome sequencing is a key component of precision health, allowing us to tailor treatment to individuals based on their genetics. Genome sequencing offers critical insights for early detection and management of various conditions including cancer, autism, infectious diseases like COVID-19, and genetic disorders. A decade ago, it cost $10 million to sequence a single genome, but today we can do the same for less than $1000 [1]. This has resulted in a large volume of sequencing data that poses significant computational challenges and requires novel computing solutions to keep pace [2].

Since the construction of the first human genome more than 20 years ago, most genomes have been sequenced using a linear reference, but it is impossible to encapsulate the diversity of our species in a single genome. When we use such a limited reference, we lose sequencing accuracy, and we misidentify important variants. Consequently, researchers are turning to reference graphs composed of multiple genomes. These multi-reference graphs, called pangenomes, demonstrate superior alignment accuracy compared to traditional linear sequence alignment methods [3]. The transition from linear to pangenome reference necessitates the establishment of new benchmarks to evaluate pangenome sequence alignment software performance. Understanding this new domain is crucial for guiding the design of future hardware and software genomics accelerators.

This work was funded in part by an NSF Graduate Research Fellowship

Alignment to pangenomes can be performed for short reads (50-150 base pairs long) or long reads (1,000-30,000+ base pairs long). Short reads are the cheapest and most common, but long reads are gaining popularity as sequencing costs decrease.

In this work, we make the following contributions.

1) We present an overview of the pangenome reference guided assembly pipeline, identifying popular tools in each step.
2) We describe the algorithms and compute patterns of two popular pangenome alignment tools for short and long reads respectively.
3) We profile the tools to identify bottlenecks and measure microarchitecture utilization.

The tools analyzed complement existing benchmarks [1] and will guide further computing research in the Pangenomics space.

II. Background

While linear sequence to reference alignment is robust to small errors such as SNPs and short INDELs, it cannot accurately align larger variants that may span an entire read. In the best case the aligner will fail to align such a read, but it may also map it to an incorrect location in the reference. Pangenomes solve this problem by allowing the read to align to multiple references with different structural variants. Pangenome references are usually represented in graph format with each node representing a subsequence of the reference. Directed edges connect nodes to link variants. Aligning to such a graph presents new challenges in both alignment and reference construction.

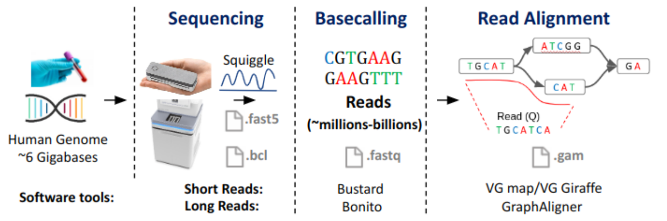

Fig. 1. Workflow for pangenome reference guided assembly. Each stage is annotated with the common tools used for short and long reads. [4]–[8]

Figure 1 outlines the workflow for aligning to a pangenome graph for both short and long reads. First, a sample is collected. The sample is then passed to a **sequencer** which reads the nucleotides and generates sequences of raw signals. In the **basecalling** process, the signals are interpreted into **reads**, sequences of base pairs in the alphabet {A, C, G, T}. The reads are then mapped onto the pangenome reference in the **alignment** process. Additionally, reads may be passed to a variant caller after alignment to identify variants present in the genome. We annotate each stage in the figure with common tools used for that task. The workflow is similar to linear alignment, but it uses different tools.

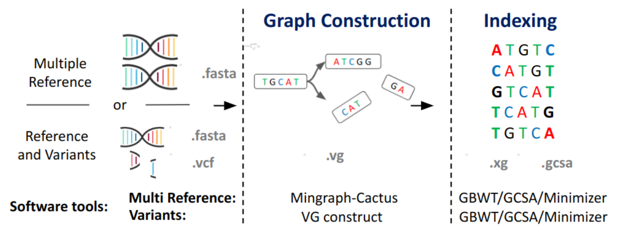

Fig. 2. Workflow for constructing a pangenome reference [6], [9]–[12]

Pangenomics also introduces a new computational workflow, namely graph construction. Figure 2 shows essential stages in constructing a pangenome reference graph annotated with common tools used for each task. The processing starts from either a set of complete reference haplotypes, or a single reference and a list of known structural variants that index that reference. The graph is indexed according to the tool that will be used for alignment. This can usually be done once for each reference, and then be used to align reads from multiple patients.

III. Pangenome Alignment Algorithms

We examine two tools specific to pangenome sequencing from the workflow shown in Figure 1, Vg Map for short reads, and GraphAligner for long reads [6], [8]. In the following subsections, we will describe the algorithmic steps in each of the tools.

A. *Vg Map*

Vg Map [6] is a short read alignment tool that can be broken down into four algorithmic steps: seeding, clustering, cluster filtering, and seed/cluster extension. It uses a graph FM-Index, Generalized Compressed Suffix Array (GCSA) to seed reads [11], and clusters them with an approximate distance metric. Next, it filters out clusters that are too small or have significant overlap with a larger cluster and are therefore not worth extending. Every pair of clusters must be considered for a total of $O(n^2)$ comparisons. For each comparison, the algorithm sorts the seeds in the cluster and finds the intersection of the two seed sets. Next it extends seeds using a custom variant of the Smith-Waterman algorithm called Graph SIMD Smith-Wateramn (GSSW) [13].

Smith-Waterman is a common extension algorithm for linear sequence alignment [14]. It operates on an $n \times m$ grid where m is the length of the query, and n is the length of a substring of the reference, generally within a constant factor of m. It

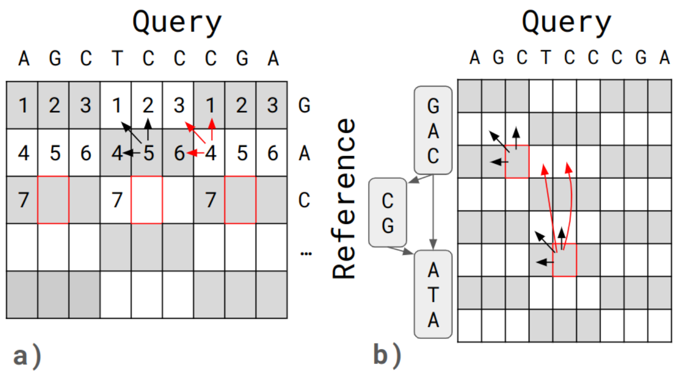

Fig. 3. Illustration of the Graph Simd Smith-Waterman (GSSW) algorithm. The dynamic programming arrays pictured have the reference string on the i-axis, and the query string on the j-axis. A cell indexed at (i, j) would hold the score associated with matching characters i and j from the reference and query respectively. **a)** The compute pattern of Farrar style strip mined Smith-Waterman. Grey shading indicates tiles, and the numbers in each cell indicate the timestep in which those cells would be computed. The red boxes indicate the cells to be computed in the next time step. The black arrows show a typical dependency pattern where the dependencies are computed before the dependent. The red arrows show a case at the end of a tile where dependencies are not precomputed. **b)** In the graph version of Simd Smith-Waterman, the reference consists of a topological sort of nodes. Dependencies within a node (the upper cell with black arrows) are handed the same as before, but dependencies at the head of a node require looking backward to consider dependencies from multiple parent nodes. This is shown in the lower node with red arrows to its parent cells.

populates this grid with alignment scores which are calculated as a function of the left, diagonal, and upper cells, i.e. cell (i, j) is dependent on cells $(i, j-1)$, $(i-1, j-1)$, and $(i-1, j)$. The algorithm can be extended to SIMD by strip mining across a row [15]. This is illustrated in figure 3.a. Each row is divided into w tiles, where w is the simd width of the processor (3 in this example). The tiles are shown with alternating grey shading. The numbers in the cells correspond to the order in which they are computed. For example, each cell with a 1 is computed in the first timestep in a 3-width simd word. Next the cells with 2, and so on. The red boxes indicate the cells to be computed in the next timestep. The black arrows show a dependency in the Smith-Waterman algorithm. In this case, every dependency has been computed before the element computed in step 5. However, this scheme leaves some unresolved dependencies between tiles. The red arrows at the beginning of a new tile show a dependency from time 4 to time 6. To resolve this dependency, Simd Smith-Waterman speculates that the element from time 6 will not contribute to the computation. Mispeculation is detected at time 6 when the dependency is computed and can be recovered by recomputing the speculative cells.

Thus far we have described a linear seed extension algorithm used for read alignment to linear references (one genome). To extend this to graphs for pangenomes, Vg Map uses an approach similar to Partial Order Alignment (POA) [16]. It uses SIMD Smith-Waterman, but instead of a linear reference on the i-axis, it aligns to a topological sort of a subgraph of the reference with directed edges between nodes, where each node corresponds to a range of rows in the graph, Figure 3. Cells within the body of a node in the dynamic programming matrix have the same dependencies they would have in SIMD Smith-Waterman, but cells in the first row of a node may have dependencies on multiple parents. This is shown in Figure 3.b with red arrows. Fortunately, because of the topological sort, these dependencies will always be computed before they are needed. In this way, Vg Map seed extension alternates between dense SIMD regions and indirect graph access dependencies.

B. *GraphAligner*

GraphAligner operates on long reads [8]. It uses a fast, simple minimizer indexing scheme [12], followed by distance-based clustering, and a less accurate extension algorithm based on Myers bitvector [17]. These algorithms make GraphAligner a poor candidate for short reads [7], but it has been shown to outperform Vg Map in accuracy for long reads, and achieves a factor of 10 speedup [8]. The bottleneck in the GraphAligner algorithm is bitvector extension.

Myers bitvector extension, like Smith-Waterman, operates by filling in a dynamic programming matrix, but it does not support the more sophisticated affine gap scoring method used in Smith-Waterman. This allows Myers to store rows as bitvectors, and break the dependencies between rows, which enables them to execute in parallel. Since the row is encoded as a bitvector, the SIMD width is also much greater than with GSSW. The algorithm can be expanded to graphs with a similar approach to POA where rows may have multiple dependencies depending on the shape of the graph [18]. To compute a new row, R_i, all parent rows are first merged via an element-wise minimum to produce a new row, R_m. R_m can then be used to compute R_i as in regular Myers bitvector. Since graphs may include cycles, it is possible that the row computed, R_i, may change the value of its parent R_p. However, any changes are strictly increasing in score. To handle such cyclic dependencies the algorithm iterates through rows with changes until the solution stabilizes.

IV. Datasets

We use Vg to construct a reference graph from a linear sequence and vcf file. The linear sequence, GRCh38, is provided by the Genome Reference Consortium and hosted by the National Center for Biotechnology Information [19] [20]. The variants are generated with minigraph-cactus by the Human Pangenome Reference Consortium [21] [22]. We limit our analysis to chromosome 22 of this reference to keep the memory requirements manageable.

We evaluate Vg Map on short reads from the human sample SRR7733443 [23]. We align the reads to a linear reference using bwa-mem2 [23] to generate a dataset of 4,457,496 reads from chromosome 22, each 151 base pairs long.

To evaluate GraphAligner, we simulate reads using pbsim2 [24] to generate one million long reads ranging from 500 to 50,000 base pairs in length sampled from the prebuilt minigraph-cactus Grch38 HPRC reference graph.

V. Performance Results

For each tool, we present three metrics:

1) End-to-end walltime of the full application, and the percentage of time spent on alignment.
2) A timing breakdown of the tool into the algorithmic stages described in section III. As these tools are threaded over reads, we report the results in terms of CPU time, the time a core spends executing the region of interest. We collect these statistics with Intel Vtune.
3) A overview of microarchitectural bottlenecks of the tool collected using Intel Vtune's Microarchitecture analysis on the alignment portion of the tool.

The specifications of the system used to collect the data are recorded in table I.

CPU	Intel Xeon Gold 6326 2.90 GHz; 32 threads per socket; 2 sockets
L1 I cache	32Kb, 8-way
L1 D cache	48Kb, 12-way
L2 cache	1.3Mb, 20-way
L3 cache	24Mb, 12-way
Memory	8 × 16GB DDR4 2933

TABLE I
System Configuration.

A. *Vg Map*

The end-to-end wall time of running Vg Map with 64 threads on the short read dataset is 271s. Of that time, 96% is spent on alignment. The remainder is almost entirely the time of loading the index (10s).

We further subdivide the time spent in alignment into algorithmic stages identified in section III-A, Figure 4. We find that cluster filtration and seed extension are the major bottlenecks for this application. Together, these kernels account for 79.2% of the alignment time. Seeding and clustering make up a small fraction of the remaining time, with the rest divided among many smaller functions, including i/o operations and access to the graph structure to extract a localized reference subgraph for alignment.

We also collect microarchitecture information on the full short-read dataset in the alignment region. Results are shown in Figure 5. [1] The application is 24.8% bounded by the front end. This is expected as most bottlenecks are in compute-bound kernels, seed extension, and cluster filtering. The observed CPI is 0.634.

B. *GraphAligner*

Running on the long read dataset with 64 threads, GraphAligner has a wall time of 4004s. This is substantially more than Vg Map because the read set is much larger. Of that time 98% is spent on alignment. Unlike Vg Map, GraphAligner does not use a cached graph index. This leads to a little more overhead, but it is a constant cost that is amortized with many reads. The code could also be redesigned to use a cached index similar to Vg Map.

[1]The segments of the piechart do not add up to exactly 100% due to sampling inaccuracy with VTune. Mispeculation is measured at 0.0% and therefore not pictured.

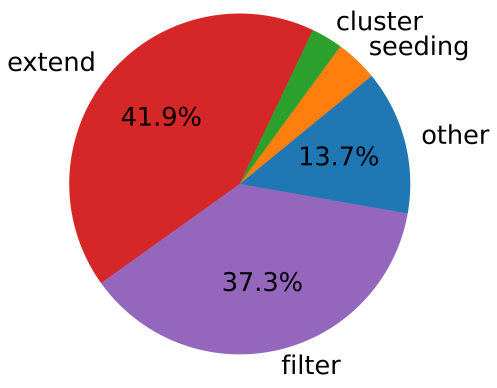

Fig. 4. Timing breakdown of alignment in vg among the following kernels: **seeding 4.03%)** finding seeds in the Graph using a graph FM-Index (GCSA), **cluster 2.99%)** grouping seeds into clusters based on distance in the genome, **filter 37.34%)** dropping clusters that are too short or overlap a lot with other clusters, **extend 41.94%)** Graph Simd Smith-Waterman cluster extension. The remaining runtime includes some i/o, and graph accesses to set up extension.

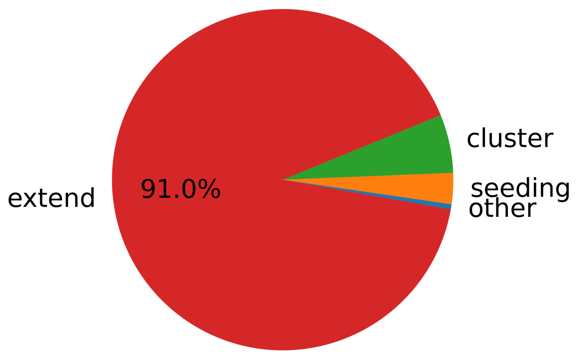

Fig. 6. Timing breakdown of alignment in GraphAligner among the following kernels: **seeding 2.94%)** finding seeds in the graph using a minimizer lookup, **cluster 5.53%)** grouping seeds by distance in the graph **extension 91.05%)** Myers bitvector cluster extension.

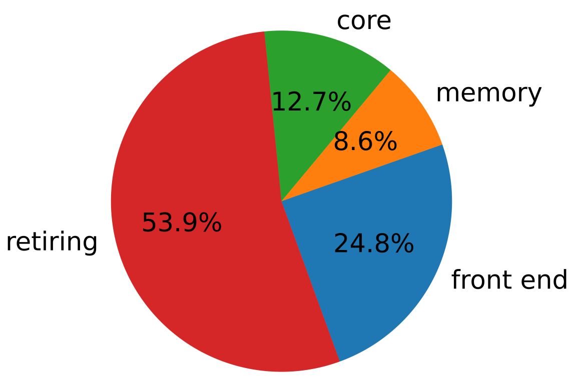

Fig. 5. Vg Map microarchitecture utilization.

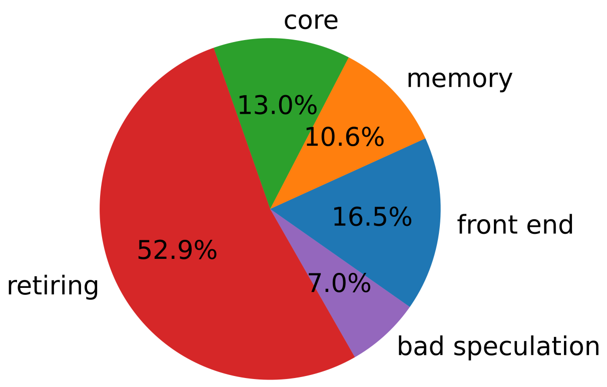

Fig. 7. GraphAligner microarchitecture utilization.

Figure 6 illustrates the timing breakdown of alignment in GraphAligner. Compared to Vg Map, GraphAligner spends much more time in seed extension. Although its extension algorithm is faster, the size of the dynamic programming matrix that must be computed grows as the square of the length of the read. Since GraphAligner is designed for and evaluated on long reads, this may explain the seed extension bottleneck. It also uses a simpler minimizer-based seeding method and does not have a costly cluster filtration step which can reduce the number of seed extensions.

Because of the size of the collected data, we profile GraphAligner's microarchitecture utilization on a subset of 10,000 reads. The results are shown in figure 7. Multiple resource constraints bound the application's performance, but no single bottleneck is evident. Overall it achieves good utilization with a CPI of 0.738.

VI. Conclusion

Advancement in genome sequencing technology generates vast amounts of data and places demand on sequencing algorithms, which can be met with custom hardware solutions. To understand genomics applications, and evaluate our designs, we need an understanding of common, compute-intensive genomics applications that bottleneck current sequencing pipelines. Previous work has collected and categorized important kernels for linear reference sequencing [1], but linear references fail to capture the diversity of variants found amongst our species. We need pangenome sequencing to improve sequencing accuracy.

We present an analysis of the pangenomics pipeline, and profile some of the important tools it uses to extract compute-intensive kernels with new characteristics not found in linear sequencing.

References

[1] A. Subramaniyan, Y. Gu, T. Dunn, S. Paul, M. Vasimuddin, S. Misra, D. Blaauw, S. Narayanasamy, and R. Das, "Genomicsbench: A benchmark suite for genomics," in *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2021, pp. 1–12.

[2] Y. Gu, A. Subramaniyan, T. Dunn, A. Khadem, K. Chen, S. Paul, M. Vasimuddin, S. Misra, D. Blaauw, S. Narayanasamy, and R. Das, "Gendp: A framework of dynamic programming acceleration for genome sequencing analysis," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*. ACM, 2023, pp. 1–15.

[3] W. Liao, M. Asri, J. Ebler, D. Doerr, M. Haukness, G. Hickey, S. Lu, J. Lucas, J. Monlong, H. Abel, and S. Buonaiuto, "A draft human pangenome reference," *Nature*, vol. 617, no. 7960, pp. 312–324, 2023.

[4] C. Oxford *et al.*, "A comparison of base-calling algorithms for illumina sequencing technology," *dated Oct*, vol. 5, p. 10, 2015.

[5] "Bonito," accesed:16-Jun-2024. [Online]. Available: https://github.com/nanoporetech/bonito

[6] E. Garrison, J. Sirén, A. M. Novak, G. Hickey, J. M. Eizenga, E. T. Dawson, W. Jones, S. Garg, C. Markello, M. F. Lin *et al.*, "Variation graph toolkit improves read mapping by representing genetic variation in the reference," *Nature biotechnology*, vol. 36, no. 9, pp. 875–879, 2018.

[7] J. Sirén, J. Monlong, X. Chang, A. M. Novak, J. M. Eizenga, C. Markello, J. A. Sibbesen, G. Hickey, P.-C. Chang, A. Carroll *et al.*, "Pangenomics enables genotyping of known structural variants in 5202 diverse genomes," *Science*, vol. 374, no. 6574, p. abg8871, 2021.

[8] M. Rautiainen and T. Marschall, "Graphaligner: rapid and versatile sequence-to-graph alignment," *Genome biology*, vol. 21, no. 1, p. 253, 2020.

[9] G. Hickey, J. Monlong, J. Ebler, A. M. Novak, J. M. Eizenga, Y. Gao, T. Marschall, H. Li, and B. Paten, "Pangenome graph construction from genome alignments with minigraph-cactus," *Nature biotechnology*, pp. 1–11, 2023.

[10] J. Sirén, E. Garrison, A. M. Novak, B. Paten, and R. Durbin, "Haplotype-aware graph indexes," *Bioinformatics*, vol. 36, no. 2, pp. 400–407, 2020.

[11] J. Sirén, "Indexing variation graphs," in *2017 Proceedings of the ninteenth workshop on algorithm engineering and experiments (ALENEX)*. SIAM, 2017, pp. 13–27.

[12] M. Roberts, W. Hayes, B. R. Hunt, S. M. Mount, and J. A. Yorke, "Reducing storage requirements for biological sequence comparison," *Bioinformatics*, vol. 20, no. 18, pp. 3363–3369, 2004.

[13] M. Zhao, W.-P. Lee, E. P. Garrison, and G. T. Marth, "Ssw library: an simd smith-waterman c/c++ library for use in genomic applications," *PloS one*, vol. 8, no. 12, p. e82138, 2013.

[14] T. F. Smith, M. S. Waterman *et al.*, "Identification of common molecular subsequences," *Journal of molecular biology*, vol. 147, no. 1, pp. 195–197, 1981.

[15] M. Farrar, "Striped smith–waterman speeds database searches six times over other simd implementations," *Bioinformatics*, vol. 23, no. 2, pp. 156–161, 2007.

[16] C. Lee, C. Grasso, and M. F. Sharlow, "Multiple sequence alignment using partial order graphs," *Bioinformatics*, vol. 18, no. 3, pp. 452–464, 2002.

[17] G. Myers, "A fast bit-vector algorithm for approximate string matching based on dynamic programming," *Journal of the ACM (JACM)*, vol. 46, no. 3, pp. 395–415, 1999.

[18] M. Rautiainen, V. Mäkinen, and T. Marschall, "Bit-parallel sequence-to-graph alignment," *Bioinformatics*, vol. 35, no. 19, pp. 3599–3607, 2019.

[19] V. A. Schneider, T. Graves-Lindsay, K. Howe, N. Bouk, H.-C. Chen, P. A. Kitts, T. D. Murphy, K. D. Pruitt, F. Thibaud-Nissen, D. Albracht *et al.*, "Evaluation of grch38 and de novo haploid genome assemblies demonstrates the enduring quality of the reference assembly," *Genome research*, vol. 27, no. 5, pp. 849–864, 2017.

[20] "Genome assembly grch38.p14reference," accessed: 16-Jun-2024. [Online]. Available: https://www.ncbi.nlm.nih.gov/datasets/genome/GCF_000001405.40/

[21] W.-W. Liao, M. Asri, J. Ebler, D. Doerr, M. Haukness, G. Hickey, S. Lu, J. K. Lucas, J. Monlong, H. J. Abel *et al.*, "A draft human pangenome reference," *Nature*, vol. 617, no. 7960, pp. 312–324, 2023.

[22] "Human pangenome reference consortium s3 bucket," accessed: 16-Jun-2024. [Online]. Available: https://s3-us-west-2.amazonaws.com/human-pangenomics/index.html?prefix=pangenomes/freeze/freeze1/minigraph-cactus/hprc-v1.1-mc-grch38/

[23] M. Vasimuddin, S. Misra, H. Li, and S. Aluru, "Efficient architecture-aware acceleration of bwa-mem for multicore systems," in *2019 IEEE international parallel and distributed processing symposium (IPDPS)*. IEEE, 2019, pp. 314–324.

[24] Y. Ono, K. Asai, and M. Hamada, "Pbsim2: a simulator for long-read sequencers with a novel generative model of quality scores," *Bioinformatics*, vol. 37, no. 5, pp. 589–595, 2021.