Evaluating the Suitability of Minimap2 for PIM Architectures

Abstract

Motivation. With the emergence of third-generation sequencing, mapping long reads against a reference genome has become a fundamental task for performing genomic analysis. This motivates the search for better algorithms and methods to devise new tools or improve existing ones. In this work, we focus on profiling the widely used program Minimap2 to understand its bottlenecks and evaluate whether it could benefit from processing-in-memory (PIM) accelerators.

Results. Using several long-read generators, we show that the Minimap2 chaining stage becomes the most prevalent bottleneck when mapping reads with low divergence from the reference genome. The chaining algorithm has sequential constraints and thus could hardly fully leverage a massively parallel scheme. Besides, the micro-instruction pipeline does not exhibit particular stalls regarding memory management. We conclude that the Minimap2 seed-chain-align paradigm may not be the best fit for an implementation on the UPMEM PIM architecture.

Table of Contents

1 Introduction

Third-generation sequencing produces long reads of 15+ kb or even ultra-long reads of 100+ kb. As the technology progresses, these reads get more accurate and constitute invaluable data for genomics research. Reads of this length need specialized tools, as many algorithms devised for next-generation sequencing (i.e., short reads of ~100 base pairs) are not able to scale. We focus on Minimap2, a state-of-the-art alignment program able to map long reads against a large reference genome. This tool being widely used among the genomics community motivates the search for improvements to accelerate the processing even further. Existing works focus on executing certain stages of the algorithm on accelerators such as GPUs or FPGAs. We broaden the scope by considering Processing-in-memory (PIM) architectures, in particular the near-memory model from UPMEM [1].

In this report, we perform some profiling of Minimap2 to pinpoint its bottlenecks and determine whether it could constitute a good candidate to be ported to PIM architectures. To this end, we use several long-read generators to simulate Oxford Nanopore (ONT) or PacBio HiFi (PB) reads of varying accuracy.

First, we give some background on the Minimap2 algorithm and the existing literature improving it. Then, we present our setup for profiling the Minimap2 program. We give our results in Section 4 and finally share our conclusions in Section 5.

2 Background

Minimap2 [6] is a widely used open-source alignment program[1]. Built on top of its predecessor [5], it uses a seed-chain-align approach. First, it creates an index of the reference database by storing minimizer positions in a hash table. Then, it collects the minimizers of a query sequence and finds exact matches (called anchors) with the index. Next, it identifies a colinear set of anchors (called chains) through dynamic programming (DP). Finally, it applies DP to perform global alignment between adjacent anchors within a chain.

Over the years, the Minimap2 program has been enriched with many improvements to handle diverse input types and increase mapping quality [7]. It has become a state-of-the-art alignment tool for long reads, and many of its core ideas, in particular the chaining algorithm and concave gap functions, are reused by novel tools [10]. Many recent works also focus on accelerating Minimap2. For instance, [4] uses SIMD parallelization to optimize all three stages of the algorithm. Others, such as [2] and [9], determine with profiling that chaining is the main bottleneck of Minimap2 and thus explore how to accelerate this particular step on FPGA and GPU. In this report, we extend their profiling analysis to characterize more precisely the bottlenecks of Minimap2.

3 Profiling Setup

We use the latest version (v2.26) of Minimap2. We instrument the code to add timers and compute the elapsed time in each stage of the program: index generation, seeding, chaining and alignment. To get accurate timing measures without introducing mutexes between queries, we run the program with only one thread. Additionally, we use Intel® VTune™ Profiler 2023.2.0 to analyze the micro-architecture pipeline of the program. We run all our tests on the same machine with an Intel® Core™

[1] `https://github.com/lh3/minimap2`

i7-8650U 4 cores @ 4.2 GHz processor and 32 GB of DDR4 @ 2.4 GHz, running Fedora Workstation 37.

Our code instrumentation, in addition to our scripts for generating datasets and producing results, is publicly available in our fork of Minimap2[2].

3.1 Reference Genome

As a reference genome, we consider human chromosome No. 2 from the February 2009 human reference sequence (GRCh37)[3]. It contains approximately 243 million non-ambiguous bases.

3.2 Long Reads Generation

We consider two long-read generators:

- *Badread* [12][4] trades some realism against giving users more control over the number and types of errors present in reads. With the right settings, it can imitate both ONT and PB data.
- *SimLoRD* [11][5] focuses on PB reads and provides some parameters to control the number of sequencing errors.

We generate datasets by sampling from our reference genome (i.e., human chromosome No. 2), except for one where we instead sample from chromosome No. 3. The latter will simulate a scenario where we try to map reads against what turns out to be the wrong database. We vary the amount of errors and produce different types of datasets labeled from *worse* and *bad* (many errors) to *good* and *best* (high accuracy). Also, we use different lengths, from 35 kb to 110 kb, to have both long and ultra-long read datasets. The exact commands and parameters used are given in our public code repository to allow for reproducing our analysis.

4 Results

First, we focus on the relative contribution of each step of Minimap2 to the elapsed time. Then, we try to understand what causes each step to be a bottleneck or not, depending on the dataset. Finally, we complete our analysis with VTune to see where the micro-instruction pipeline stalls.

4.1 Contribution of each step

Indexing the reference genome is by far the longest step in Minimap2. We measure that it takes approximately 8.7 seconds to index human chromosome No. 2 (~243 Mb). The tool, however, gives the possibility of saving the index on the disk for later reuse. The index construction is thus a one-time cost, and we decide to focus on the three other stages in the remainder of this analysis.

We see in Figure 1 the contribution of the three other steps for each query dataset. We notice an interesting phenomenon. Alignment takes the most time when there are many errors, while very

[2] https://github.com/fdemoor2/minimap2-instrument
[3] http://hgdownload.cse.ucsc.edu/goldenPath/hg19/chromosomes
[4] https://github.com/rrwick/Badread
[5] https://bitbucket.org/genomeinformatics/simlord

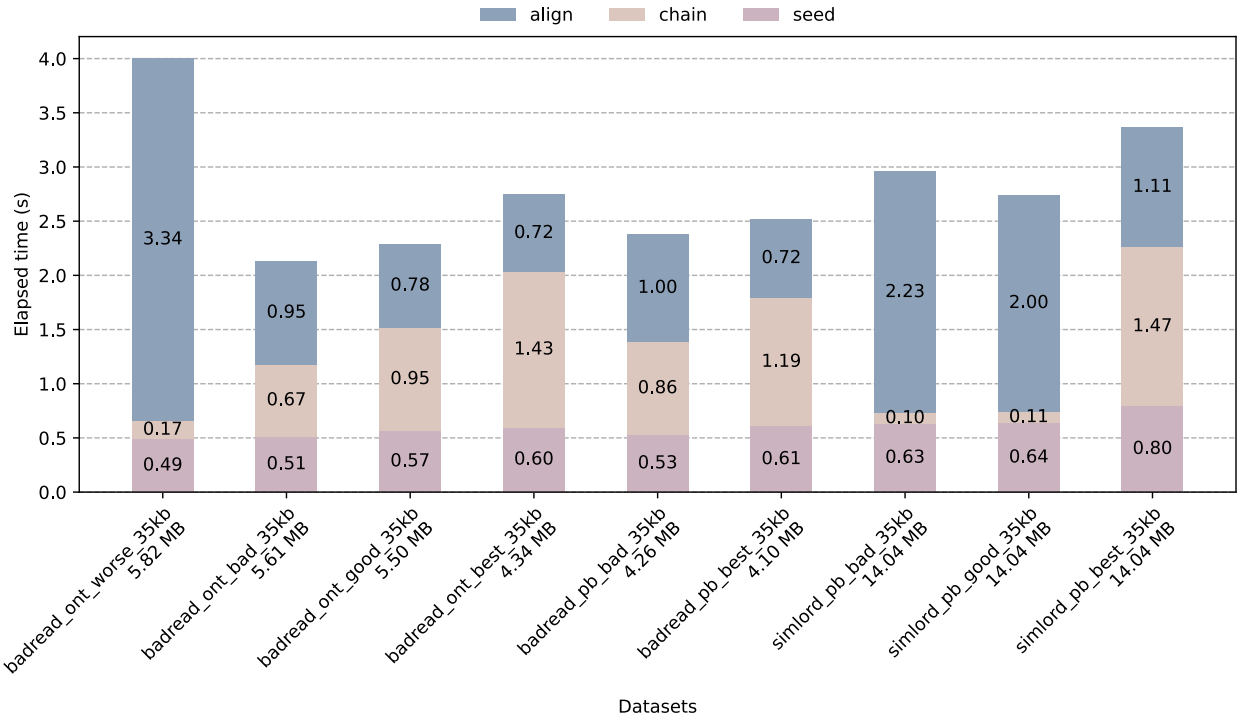

Figure 1: Chaining becomes the main bottleneck for accurate reads

accurate reads expose chaining as the main bottleneck. This result is actually quite intuitive. With accurate reads, the chaining does most of the work and finds long primary chains. There is little work left to do to complete the mapping in the alignment stage. The opposite happens with noisy reads: the chaining will find chains with smaller coverage overall, and the alignment stage has much more space to fill. As the technology progresses, it is reasonable to expect sequencing to get more accurate. The chaining should thus become the most prevalent bottleneck in Minimap2.

We show in Figure 2 the timing breakdown for datasets of varying read sizes. We see that the higher read size increases the chaining part regarding the very accurate ONT reads generated with Badread. The program is probably able to build better and longer chains in these cases. The same phenomenon is also visible on the noisy reads, but less pronounced. However, the SimLoRD datasets do not show the same behavior. This might be a distinctive point between ONT and PB technologies, which do not exhibit the same types of sequencing issues. One main difference is, for instance, that PB errors are randomly distributed while ONT errors (which include both signal measurements and basecalling) are biased [14].

4.2 Correlation with the Mapping Divergence

To better understand what makes the chaining or the alignment become the bottleneck, we consider the divergence score Minimap2 outputs for each mapping. This measures how accurately the query matches the reference genome. For each dataset, we compute the box plot of the divergence scores of all mappings obtained and show the results in Figure 3. Interestingly, we see a clear correlation between the median divergence score and the timing breakdown. Noisy reads output a high divergence, and the execution is stalled by alignment. Accurate reads give a low divergence, and the program spends more time in the chaining stage. We also see that mapping the dataset

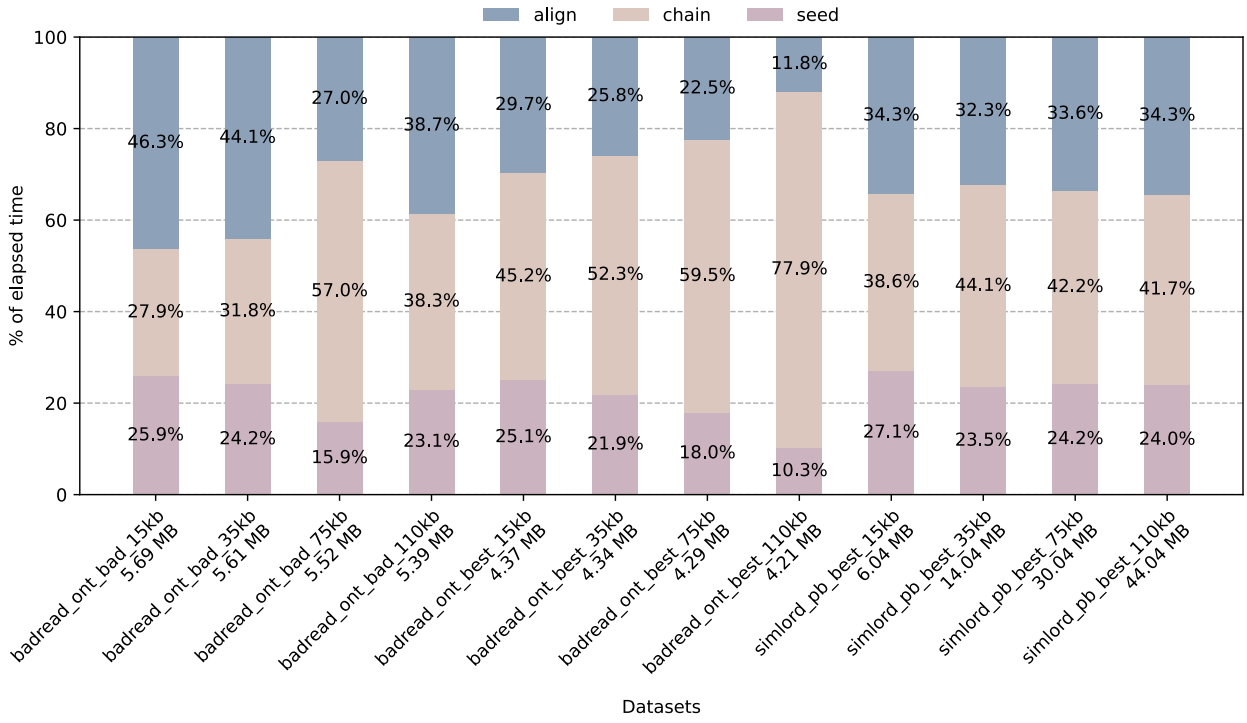

Figure 2: Impact of read size on Minimap2 bottleneck

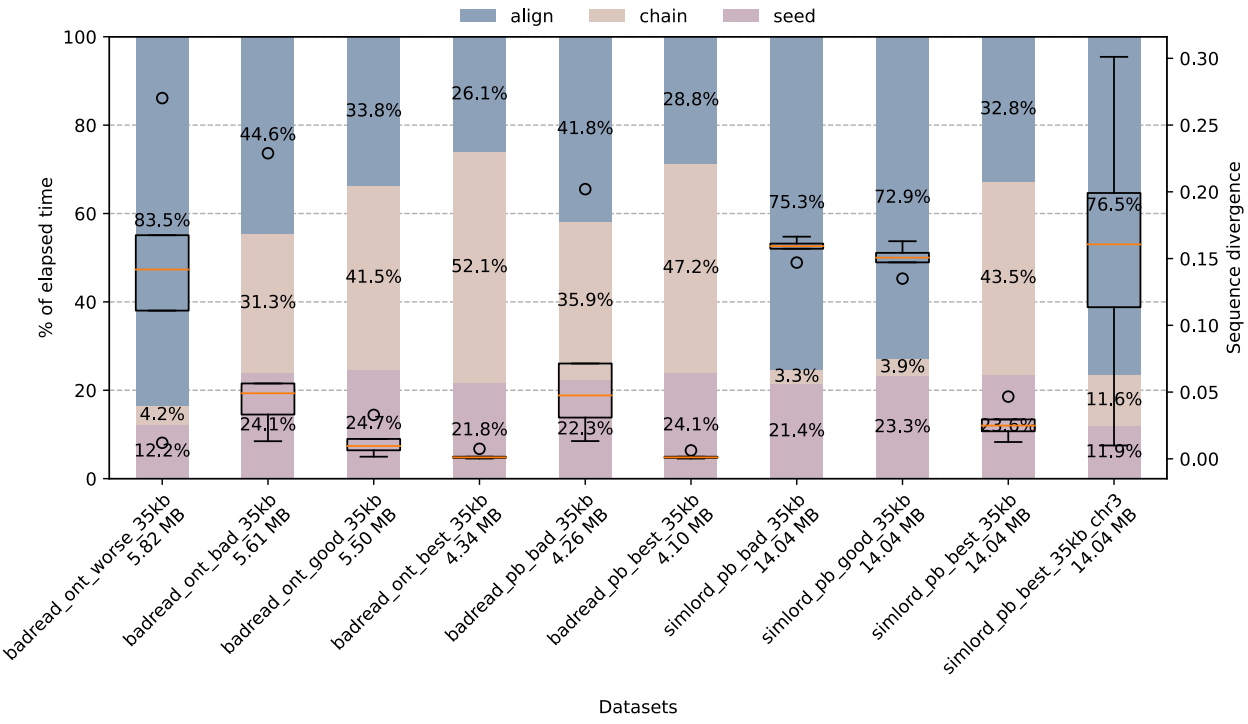

Figure 3: Alignment or chaining being the bottleneck is correlated with the result mapping divergence

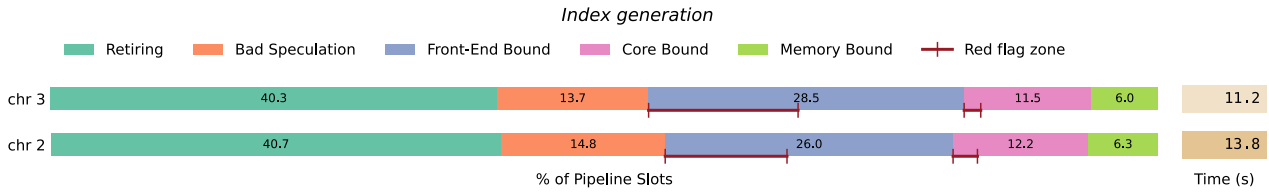

Figure 4: Indexing human chromosomes No. 2 and No. 3

sampled from chromosome No. 3 exhibits the alignment as the main bottleneck. We conclude that the chaining-alignment timing ratio is heavily correlated with how divergent the queries are from the reference database. High divergence can be caused not only by sequencing errors, but also by mapping reads against the wrong reference. The latter can happen in typical scenarios where we have unknown sequences and try to find out from which genome they come.

4.3 Analysis with VTune Profiler

To complement our analysis, we run Minimap2 with the VTune Profiler software. It is a top-down analysis tool for x86 architectures [13]. More precisely, we run the micro-architecture analysis that categorizes micro-operations depending on whether they are allocated, retired, or stalled. It indicates red-flagged metrics and allows one to spot the pipeline stalls. To present results, we plot the distribution of the following categories in terms of pipeline slot percentage: retiring, bad speculation, front-end bound, core bound and memory bound.

We first analyze the index generation, both on chromosome No. 2 and No. 3 reference genomes. We see in Figure 4 that this process is mainly front-end bound, mostly because of stalls when fetching the right instructions after branch mispredictions.

Because Minimap2 executes all three steps of the mapping with intra-read parallelism, it is tricky to isolate the contribution of each part with VTune as we would need to turn the collection on and off at a high frequency. Instead, we resort to a cumulative approach and use the built-in option to stop after the chaining and not perform base alignment. We select a few datasets among those presented in Section 3 and map them against the chromosome No. 2 reference genome. We give results for the seeding and chaining steps in Figure 5 and for the full mapping in Figure 6. Plots show some significant bad speculation during the chaining, which probably takes root in the chaining score optimization problem solved through DP. The algorithm indeed tries to find the best preceding anchor, which implies some branching to discard anchors too far away and find the local optimum. This step is also core-bound.

The alignment stage, however, does not seem to exhibit particular stalls, as all scores except the retiring percentage decrease from the first plot to the second. Overall, we see that the tool is not memory-bound and is quite well optimized. This confirms that Minimap2 is a mature program and explains its broad usage in research and production environments.

5 Discussion and Conclusion

In this work, we performed a profiling analysis of the Minimap2 alignment tool. We found a clear correlation between the resulting mapping divergence and the main bottleneck of the algorithm. Querying sequences very divergent from the genome reference exposes the alignment part as the major

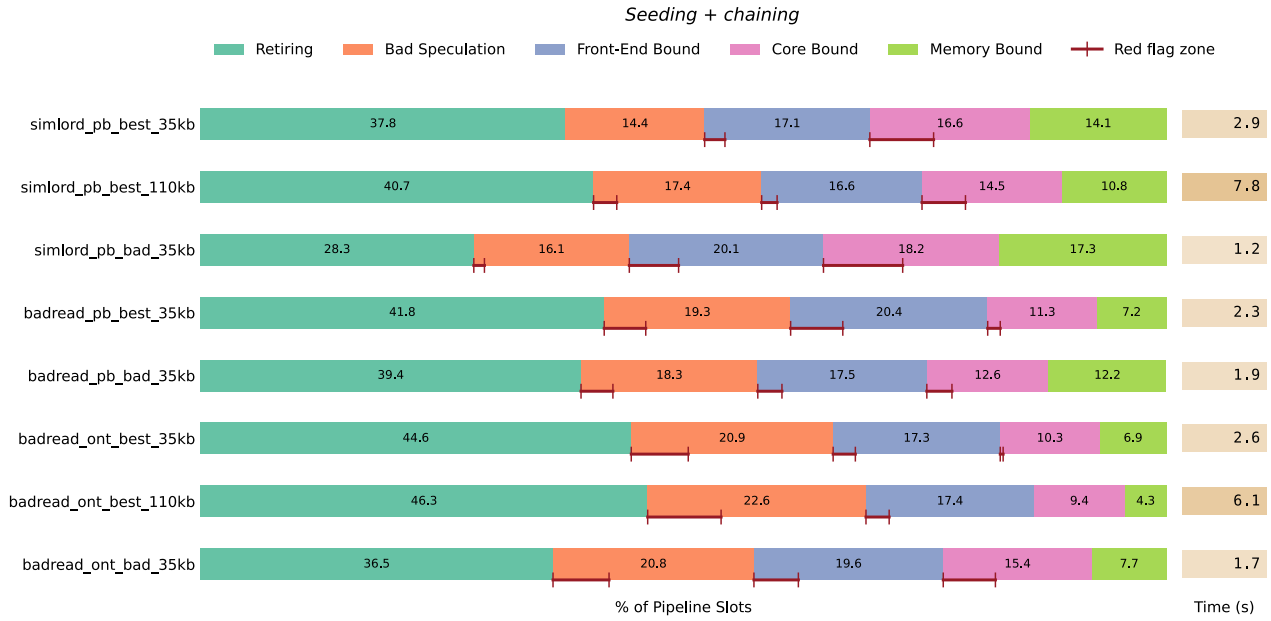

Figure 5: Executing seeding and chaining steps

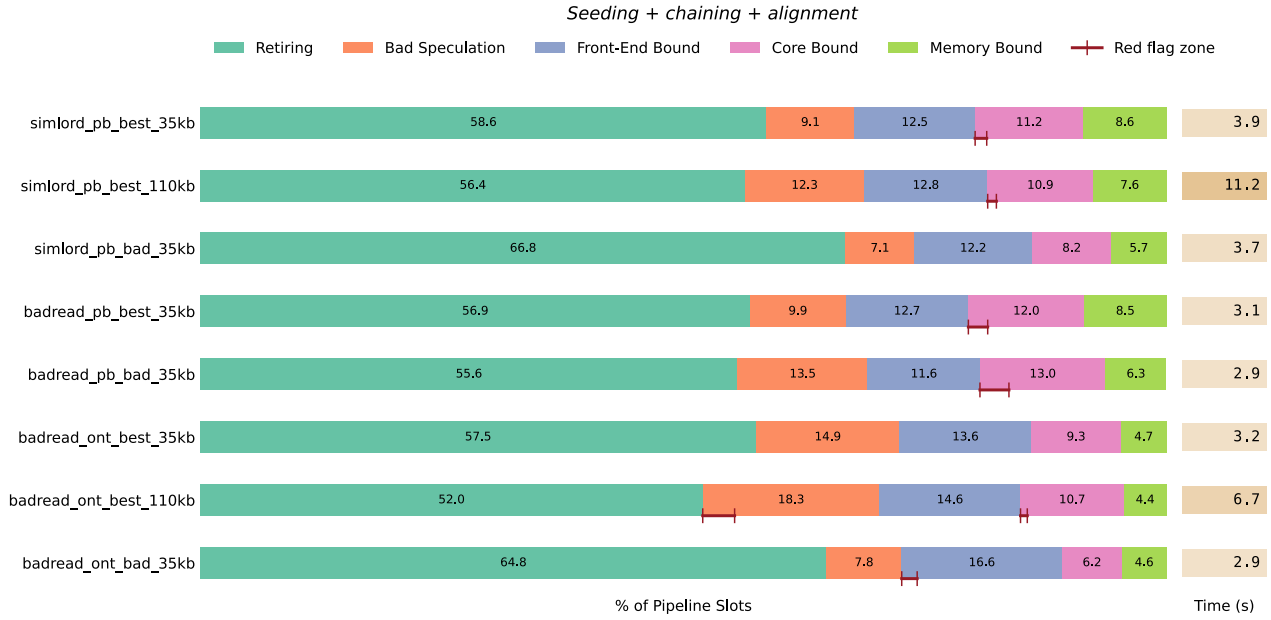

Figure 6: Executing the full mapping from seeding to alignment

stalling step. On the contrary, lightly divergent queries cause the program to spend more time in the chaining process. Overall, the micro-architecture pipeline has some front-end and bad speculation stalls, but does not expose significant memory bound issues. The latter constitutes a first clue that the Minimap2 algorithm might not be a good fit for the PIM architectures. Memory bound workloads are indeed what benefit the most from the strengths of this hardware [3].

Moreover, the ongoing advances in third-generation sequencing technologies make long reads even more accurate. For instance, the latest Nanopore R10.4 flow cells outperform their predecessors, R9.4, in terms of read accuracy [8]. Unless we try to map reads of unknown origin against a large set of reference genomes, the chaining step of Minimap2 is expected to become the most prevalent bottleneck, which could limit researchers from making important new discoveries. The chaining stage is, however, poorly suited for parallelism because of the sequential nature of its score optimization problem. Existing work obtains interesting speedups by reordering computations and performing this step on FPGAs or GPUs [2, 9]. Yet, we do not believe such implementations could be easily adapted to the UPMEM PIM architecture. The inherent sequential constraints of the chaining algorithm make it difficult to fully leverage this massively parallel programming environment and limit our hopes for significant acceleration. Improvements could be obtained for the index generation and seeding phases, but our analysis shows these stages are less relevant to optimize.

Overall, we believe the Minimap2 program may not be a good candidate for a PIM implementation. However, other long-read alignment programs that rely on a different approach than the seed-chain-align paradigm might constitute better fits. Extending this analysis with more state-of-the-art tools will thus be an interesting future avenue of research.

REFERENCES

[1] Fabrice Devaux. The true Processing In Memory accelerator. In *2019 IEEE Hot Chips 31 Symposium (HCS)*, pages 1–24, August 2019. ISSN: 2573-2048.

[2] Licheng Guo, Jason Lau, Zhenyuan Ruan, Peng Wei, and Jason Cong. Hardware Acceleration of Long Read Pairwise Overlapping in Genome Sequencing: A Race Between FPGA and GPU. In *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 127–135, April 2019. ISSN: 2576-2621.

[3] Juan Gómez-Luna, Izzat El Hajj, Ivan Fernandez, Christina Giannoula, Geraldo F. Oliveira, and Onur Mutlu. Benchmarking Memory-Centric Computing Systems: Analysis of Real Processing-In-Memory Hardware. pages 1–7. IEEE Computer Society, October 2021.

[4] Saurabh Kalikar, Chirag Jain, Vasimuddin Md, and Sanchit Misra. Accelerating long-read analysis on modern CPUs, February 2022. Pages: 2021.07.21.453294 Section: New Results.

[5] Heng Li. Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. *Bioinformatics*, 32(14):2103–2110, July 2016.

[6] Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18):3094–3100, September 2018.

[7] Heng Li. New strategies to improve minimap2 alignment accuracy. *Bioinformatics*, 37(23):4572–4574, December 2021.

[8] Ying Ni, Xudong Liu, Zemenu Mengistie Simeneh, Mengsu Yang, and Runsheng Li. Benchmarking of Nanopore R10.4 and R9.4.1 flow cells in single-cell whole-genome amplification

and whole-genome shotgun sequencing. *Computational and Structural Biotechnology Journal*, 21:2352–2364, March 2023.

[9] Harisankar Sadasivan, Milos Maric, Eric Dawson, Vishanth Iyer, Johnny Israeli, and Satish Narayanasamy. Accelerating Minimap2 for Accurate Long Read Alignment on GPUs. *Journal of biotechnology and biomedicine*, 6(1):13, 2023. Publisher: NIH Public Access.

[10] Kristoffer Sahlin, Thomas Baudeau, Bastien Cazaux, and Camille Marchet. A survey of mapping algorithms in the long-reads era. *Genome Biology*, 24(1):133, June 2023.

[11] Bianca K. Stöcker, Johannes Köster, and Sven Rahmann. SimLoRD: Simulation of Long Read Data. *Bioinformatics*, 32(17):2704–2706, September 2016.

[12] Ryan R. Wick. Badread: simulation of error-prone long reads. *Journal of Open Source Software*, 4(36):1316, April 2019.

[13] Ahmad Yasin. A Top-Down method for performance analysis and counters architecture. pages 35–44, March 2014.

[14] Haowen Zhang, Chirag Jain, and Srinivas Aluru. A comprehensive evaluation of long read error correction methods. *BMC Genomics*, 21(6):889, December 2020.