POSTER: FastBWA: Practical and Cost-Efficient Genome Sequence Alignment Pipeline

Zhonghai Zhang[1,2], Yewen Li[1,2], Ke Meng[1], Chunming Zhang[1], Guangming Tan[1,2]
[1] State Key Lab of Processors, Institute of Computing Technology, CAS, Beijing, China
[2] University of Chinese Academy of Sciences, Beijing, China
{zhangzhonghai,liyewen19s,mengke,zhangchunming,tgm}@ict.ac.cn

Abstract

Sequence alignment is a fundamental and often time-consuming step in genomic data analysis. Typically, it adheres to the seed-and-extension paradigm and numerous accelerator-based approaches have been proposed to optimize either of the kernels. However, these approaches often increase costs and contribute minimally to the overall alignment process. To address this, we have designed an optimized full pipeline, FastBWA, which seeks to enhance performance while keeping costs low and explores the potential of CPU computing resources. Our implementation demonstrates that FastBWA achieves up to 2.5× and 1.8× in end-to-end alignment throughput compared to BWA-MEM and its newer version, BWA-MEM2.

CCS Concepts

• **Computing methodologies** → **Parallel algorithms**.

Keywords

Genome Sequence Alignment, BWA-MEM Acceleration, Parallel Application

1 Introduction

Sequence alignment is an essential building block of genomic data analysis [13, 27], which involves mapping sequence fragments (reads) to the reference genome. In essence, it is similar to finding the optimal matching position for a specific short substring within a large string. BWA-MEM [18] and its updated version, BWA-MEM2 [26], are widely used software for processing next-generation sequencing (NGS) data, following the classic seed-and-extend paradigm [16–18]. During the seeding phase, BWA-MEM employs the FM-Index search [7] algorithm to identify exact matches between small segments (seeds) of the reads and the reference. In the seed-extension phase, the Banded Smith-Waterman (BSW) algorithm [5] is used to extend both ends of the seed until the entire read is covered. Finally, matches with the highest scores are selected as the alignment results.

Numerous optimization initiatives [2, 3, 6, 8, 9, 11, 12, 14, 18–21, 24–27] have been proposed to enhance the performance of several key kernels of BWA-MEM. The Enumerated Radix Trees (ERT) [23] and BWA-MEME [14] based on learned indices introduced new seeding algorithms that take advantage of large memory consumption to reduce irregular memory access. Meanwhile, SeedEx and Darwin aim to accelerate the Smith-Waterman algorithm through hardware-based approaches. However, optimizations focused on specific kernels do not always result in substantial performance improvements for the entire process. Furthermore, large memory demands and additional hardware accelerators may increase the costs of sequence alignment. Therefore, we seek a new approach in this paper, considering the characteristics of genomic data, to find a better solution that balances performance and cost.

From a holistic perspective, we should rethink that sequence alignment does not necessarily require high-end acceleration cards for acceleration. Given that CPUs are currently more accessible and cost-effective, they may present a more promising solution. For example, in Amazon/Azure cloud, the prices of GPU and FPGA instances are about 7.32× and 5.72× that of CPU instances, respectively [1]. In this paper, we propose FastBWA, a practical cost-efficient end-to-end solution for accelerating second-generation sequencing alignment on CPU servers with the following optimizations: (i) For the seeding phase, we propose the FMT-Index data structure to eliminate irregular memory access, followed by a direct-extend strategy to leverage genomic data features to further accelerate seeding performance. (ii) For the seed-extension phase, we propose the vectorized BSW algorithm to accelerate this phase using intra-query parallelism and a sliding window mechanism on CPUs. (iii) For the whole process, we design a decoupled pipeline to alleviate the read-and-write stalls. In addition, FastBWA ensures 100% identical output to the de facto standard software BWA-MEM, making it a state-of-the-art (SOTA) version of BWA-MEM and BWA-MEM2.

2 Design of FastBWA

Seeding Phase: The design philosophy of our seeding phase is to achieve comparable performance while maintaining low memory consumption, which allows us to make our design more practical and cost-efficient. We use the *N*-step FM-Index algorithm as the basis of our design and set *N* to be two (i.e., the Ferragina-Manzini-Twice-Index, FMT-Index), which represents the optimal trade-off between single memory access throughput and memory footprint [4]. As the seed length increases, over 80% of the seeds quickly drop to one in the number of exact matching positions (hits) in the reference. At this stage, the original FM-Index algorithm still requires irregular memory accesses to extend each base. To address this, we employ a direct-extend strategy to align the remaining bases with the corresponding reference bases sequentially. This approach effectively reduces multiple memory accesses to a single access, thereby improving overall performance.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
PPoPP '25, March 1–5, 2025, Las Vegas, NV, USA
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1443-6/25/03
https://doi.org/10.1145/3710848.3710896

Seed-Extension Phase: BWA-MEM employs the Banded Smith-Waterman (BSW) [5] algorithm for seed extension. Due to significant differences in seed length and position within the reads, the lengths of the remaining sequences (queries) exhibit substantial variability. BWA-MEM2 [26] adopts an inter-query approach to parallelize the computation across multiple queries. This method suffers from low pruning efficiency since the pruning positions vary across different queries, and queries within the same vector lane must wait for the completion of the last query. Remarkably, the computational performance of this approach is even inferior to the non-parallel version of BWA-MEM under whole-exome sequencing (WES) data. We introduced an intra-query methodology that minimizes computational bubbles and can handle the pruning strategy for each query individually. Rather than performing a lookup for the location of the maximum value during the BSW computation of each individual vector lane, we update it only when a larger alignment score occurs. Fewer SIMD bubbles and redundant computations improve the overall performance of the seed-extension phase.

Pipeline Optimization: The read-and-write process becomes the performance bottleneck after optimizing the seeding phase and the seed-extension phase. To address this issue, we propose a new workflow to maximize the overlap between I/O operations and computation, thereby effectively reducing these stalls. During the sequence alignment process, various data blocks (i.e., input reads and output SAMs) are temporarily stored in memory, introducing frequent memory allocation and release, which may lead to performance loss in the multi-threading environment. BWA-MEM2 pre-allocates buffers, but they are difficult to adjust flexibly. When involved in reallocation and release operations, they can also lead to performance loss. To tackle this problem, we introduce an elastic memory allocation mechanism. This is based on an observation that in the sequence alignment process, both reads and SAMs are small (1K bytes) string sequences with different lengths. When reads are loaded for the first time, we allocate memory for each read according to its actual size. After these reads are processed, the corresponding small memories are reserved and used as independent buffers to store the reads for the next time they are loaded.

3 Evaluation

Datasets: To compare the performance of FastBWA and other baselines, We use the reference human genome assembly (human_g1k_v37 [22]) and five datasets to evaluate the execution time [10]. Table 1 shows the details of the five Homo sapiens datasets we adopted. Our datasets include three next-generation sequencing lengths (i.e., 101 bp, 144 bp, and 150 bp) and two sequencing categories (i.e., whole-genome sequencing (WGS) and whole-exome sequencing (WES)).

Table 1: Five experimental datasets of Homo sapiens [10].

Label	Type	Len.	Num.	Source
D1	WES	144 bp	90M	SRR25735653
D2	WGS	101 bp	90M	ERR194147
D3	WGS	150 bp	90M	SRR25735658
D4	WES	150 bp	90M	SRR25735654
D5	WGS	150 bp	90M	ERR9129380

Baseline: We use four SOTA baselines to compare their performance with FastBWA: BWA-MEM [18], BWA-MEM2 [26], ERT-BWA-MEM2 [23], and BWA-MEME [14][1] [21]. We use GCC 11.4 and the -O3 parameter to compile and run all the CPU baselines on an AMD Ryzen Threadripper 3970X 32-core workstation.

Results: Figure 1 shows the end-to-end execution time for all five datasets. We can observe that: (i) Compared to the other four baselines BWA-MEM [18], BWA-MEM2 [26], ERT-BWA-MEM2 [23], and BWA-MEME [14], FastBWA achieves speedups of up to 2.28×, 1.84×, 1.69×, 2.17×, and 2.94×, respectively. The average execution time speedup of FastBWA is 2.96×, as it optimizes the seeding phase, the seed-extension phase, and the system level. (ii) FastBWA has stable performance improvements on WGS and WES datasets (2.09× to 2.28× compared to BWA-MEM). In contrast, BWA-MEM2, ERT-BWA-MEM2, and BWA-MEME all perform poorly on the WES datasets. The WES datasets generate more seeds with length differences in adjacent regions, exacerbating the redundant computation problem of the inter-query SIMD implementation in the seed-extension phase.

In addition to comparing performance on the same platform, we also used Alibaba Cloud to assess costs. FastBWA and the baselines utilized hardware platforms with minimum requirements to process the same amount of data. Since FastBWA does not require large memory or additional hardware devices and achieves optimal performance under the same CPU hardware conditions, it incurs the lowest cost when processing the same amount of data. FastBWA achieves up to 2.28×, 1.84×, 2.52×, 3.23× cost reduction, respectively. This emphasizes the cost-effectiveness of FastBWA compared to baselines. These results demonstrate the efficacy of FastBWA in delivering high performance at a lower cost, making it a valuable tool for genome sequence alignment in bioinformatics.

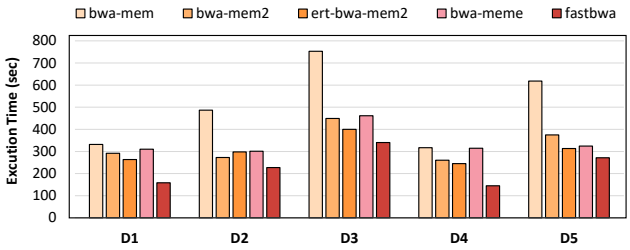

Figure 1: The performance details of BWA-MEM [18], BWA-MEM2 [26], ERT-BWA-MEM2 [23], BWA-MEME [14], and FastBWA in all five datasets.

4 Acknowledgments

We thank all anonymous reviewers for their valuable comments and helpful suggestions. This work was supported in part by National Natural Science Foundation of China(62032023) and National Science Fund for Distinguished Young Scholars(T2125013).

[1]We notice that BWA-MEM-SCALE [15] is a SOTA baseline but it can not run on our experimental platform due to program errors. Since its throughput in the first two epochs is close to that of ERT-BWA-MEM2 (about 98%), we use ERT-BWA-MEM2 to estimate its performance.

References

[1] [n. d.]. AWS EC2. https://aws.amazon.com. (Accessed on 04/03/2024).

[2] Nauman Ahmed, Jonathan Lévy, Shanshan Ren, Hamid Mushtaq, Koen Bertels, and Zaid Al-Ars. 2019. GASAL2: A GPU Accelerated Sequence Alignment Library for High-Throughput NGS Data. *BMC Bioinformatics* 20, 1 (Dec. 2019), 520. https://doi.org/10.1186/s12859-019-3086-9

[3] Mohammed Alser, Hasan Hassan, Akash Kumar, Onur Mutlu, and Can Alkan. 2019. Shouji: a fast and efficient pre-alignment filter for sequence alignment. *Bioinformatics* 35, 21 (03 2019), 4255–4263. https://doi.org/10.1093/bioinformatics/btz234

[4] Alejandro Chacón, Santiago Marco-Sola, Antonio Espinosa, Paolo Ribeca, and Juan Carlos Moure. 2014. Boosting the FM-index on the GPU: Effective techniques to mitigate random memory access. *IEEE/ACM transactions on computational biology and bioinformatics* 12, 5 (2014), 1048–1059.

[5] Kun-Mao Chao, William R. Pearson, and Webb Miller. 1992. Aligning two sequences within a specified diagonal band. *Bioinformatics* 8, 5 (1992), 481–487. https://doi.org/10.1093/bioinformatics/8.5.481

[6] Max Doblas, Oscar Lostes-Cazorla, Quim Aguado-Puig, Nick Cebry, Pau Fontova-Musté, Christopher Frances Batten, Santiago Marco-Sola, and Miquel Moretó. 2023. GMX: Instruction Set Extensions for Fast, Scalable, and Efficient Genome Sequence Alignment. *2023 56th IEEE/ACM International Symposium on Microarchitecture (MICRO)* (2023), 1466–1480. https://api.semanticscholar.org/CorpusID:266085103

[7] Paolo Ferragina and Giovanni Manzini. 2000. Opportunistic Data Structures with Applications. In *FOCS*. IEEE, 390–398.

[8] Daichi Fujiki, Arun Subramaniyan, Tianjun Zhang, Yu Zeng, Reetuparna Das, David Blaauw, and Satish Narayanasamy. 2018. GenAx: A genome sequencing accelerator. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 69–82.

[9] Daichi Fujiki, Shunhao Wu, Nathan Ozog, Kush Goliya, David Blaauw, Satish Narayanasamy, and Reetuparna Das. 2020. SeedEx: A Genome Sequencing Accelerator for Optimal Alignments in Subminimal Space. In *MICRO*. IEEE/ACM, Athens, Greece, 937–950. https://doi.org/10.1109/MICRO50266.2020.00080

[10] Lewis Y Geer, Aron Marchler-Bauer, Renata C Geer, Lianyi Han, Jane He, Siqian He, Chunlei Liu, Wenyao Shi, and Stephen H Bryant. 2010. The NCBI biosystems database. *Nucleic acids research* 38, suppl_1 (2010), D492–D496.

[11] Sneha D. Goenka, Yatish Turakhia, Benedict Paten, and Mark Horowitz. 2020. SegAlign: A Scalable GPU-Based Whole Genome Aligner. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–13. https://doi.org/10.1109/SC41405.2020.00043

[12] Sree Charan Gundabolu, T. N. Vijaykumar, and Mithuna Thottethodi. 2021. FastZ: accelerating gapped whole genome alignment on GPUs. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '21)*. Association for Computing Machinery, New York, NY, USA, Article 8, 13 pages. https://doi.org/10.1145/3458817.3476202

[13] Tae Jun Ham, David Bruns-Smith, Brendan Sweeney, Yejin Lee, Seong Hoon Seo, U Gyeong Song, Young H. Oh, Krste Asanovic, Jae W. Lee, and Lisa Wu Wills. 2020. Genesis: A Hardware Acceleration Framework for Genomic Data Analysis. In *ISCA*. ACM/IEEE, Los Angeles, CA.

[14] Youngmok Jung and Dongsu Han. 2022. BWA-MEME: BWA-MEM emulated with a machine learning approach. *Bioinformatics* 38, 9 (03 2022), 2404–2413. https://doi.org/10.1093/bioinformatics/btac137 arXiv:https://academic.oup.com/bioinformatics/article-pdf/38/9/2404/49874105/btac137.pdf

[15] Changdae Kim, Kwangwon Koh, Taehoon Kim, Daegyu Han, and Jiwon Seo. 2023. BWA-MEM-SCALE: Accelerating Genome Sequence Mapping on Commodity Servers. In *Proceedings of the 51st International Conference on Parallel Processing* (Bordeaux, France) *(ICPP '22)*. Association for Computing Machinery, New York, NY, USA, Article 83, 12 pages. https://doi.org/10.1145/3545008.3545033

[16] Heng Li. 2013. Aligning Sequence Reads, Clone Sequences and Assembly Contigs with BWA-MEM. *arXiv:1303.3997 [q-bio]* (May 2013). arXiv:1303.3997 [q-bio]

[17] H. Li and R. Durbin. 2009. Fast and Accurate Short Read Alignment with Burrows-Wheeler Transform. *Bioinformatics* 25, 14 (July 2009), 1754–1760. https://doi.org/10.1093/bioinformatics/btp324

[18] Heng Li and Richard Durbin. 2010. Fast and accurate long-read alignment with Burrows–Wheeler transform. *Bioinformatics* 26, 5 (01 2010), 589–595. https://doi.org/10.1093/bioinformatics/btp698

[19] Shuangchen Li, Dimin Niu, Yuhao Wang, Wei Han, Zhe Zhang, Tianchan Guan, Yijin Guan, Heng Liu, Linyong Huang, Zhaoyang Du, Fei Xue, Yuanwei Fang, Hongzhong Zheng, and Yuan Xie. 2022. Hyperscale FPGA-as-a-service architecture for large-scale distributed graph neural network. *Proceedings of the 49th Annual International Symposium on Computer Architecture* (2022).

[20] Yewen Li, Xueqi Li, Ruihao Gao, Wanqi Liu, and Guangming Tan. 2023. NvWa: Enhancing Sequence Alignment Accelerator Throughput via Hardware Scheduling. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 1236–1248.

[21] Minh Pham, Yicheng Tu, and Xiaoyi Lv. 2023. Accelerating BWA-MEM Read Mapping on GPUs. In *Proceedings of the 37th International Conference on Supercomputing* (Orlando, FL, USA) *(ICS '23)*. Association for Computing Machinery, New York, NY, USA, 155–166. https://doi.org/10.1145/3577193.3593703

[22] Nayanah Siva. 2008. 1000 Genomes project. *Nature biotechnology* 26, 3 (2008), 256–257.

[23] Arun Subramaniyan, Jack Wadden, Kush Goliya, Nathan Ozog, Xiao Wu, Satish Narayanasamy, David Blaauw, and Reetuparna Das. 2021. Accelerated Seeding for Genome Sequence Alignment with Enumerated Radix Trees. In *ISCA*. ACM/IEEE.

[24] Yatish Turakhia, Gill Bejerano, and William J. Dally. 2018. Darwin: A Genomics Co-processor Provides up to 15,000X Acceleration on Long Read Assembly. In *ASPLOS*. ACM, Williamsburg, VA, USA, 199–213. https://doi.org/10.1145/3173162.3173193

[25] Yatish Turakhia, Sneha D. Goenka, Gill Bejerano, and WIlliam J. Dally. 2019. Darwin-WGA: A Co-processor Provides Increased Sensitivity in Whole Genome Alignments with High Speedup. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 359–372. https://doi.org/10.1109/HPCA.2019.00050

[26] Md. Vasimuddin, Sanchit Misra, Heng Li, and Srinivas Aluru. 2019. Efficient Architecture-Aware Acceleration of BWA-MEM for Multicore Systems. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 314–324. https://doi.org/10.1109/IPDPS.2019.00041

[27] Lisa Wu, David Bruns-Smith, Frank A. Nothaft, Qijing Huang, Sagar Karandikar, Johnny Le, Andrew Lin, Howard Mao, Brendan Sweeney, Krste Asanovic, David A. Patterson, and Anthony D. Joseph. 2019. FPGA Accelerated INDEL Realignment in the Cloud. In *HPCA*. IEEE, Washington, DC, USA, 277–290. https://doi.org/10.1109/HPCA.2019.00044