G^3SA : A GPU-Accelerated Gold Standard Genomics Library for End-to-End Sequence Alignment

Yeejoo Han*
Seoul National University
Seoul, Republic of Korea
hyjoo16@snu.ac.kr

Sunwoo Kim*
Seoul National University
Seoul, Republic of Korea
kadash.sean_kim@snu.ac.kr

Seongyeon Park
Seoul National University
Seoul, Republic of Korea
syeonp@snu.ac.kr

Jinho Lee
Seoul National University
Seoul, Republic of Korea
leejinho@snu.ac.kr

Abstract

Sequence alignment is the first step of the bioinformatics pipelines for analyzing DNA sequences in genomics. As the DNA sequencer throughput is dramatically increasing, the pipeline bottleneck has now shifted to the massive amount of sequence alignment computation. GPU-based acceleration has emerged as a promising solution to this challenge, and several efforts have focused on accelerating core algorithmic components. However, when integrating existing GPU-accelerated components into BWA-MEM and Minimap2, two widely used tools for short and long reads respectively, we identify several unresolved issues. One could naively attempt to gather and/or fix those libraries, but such an approach would result in a severe slowdown despite non-trivial effort.

To address these issues, we propose G^3SA, the first GPU acceleration with efficient parallelization and optimization strategies for the end-to-end gold standard sequence alignment algorithms for short and long reads. We observe that the primary bottlenecks stem from irregular and redundant computation and memory access patterns. Therefore, we design our kernels to fully exploit the memory hierarchy and computational capabilities of GPUs, reducing costly computation and memory access. Furthermore, following the seed-chain-extend flow of standard sequence alignment algorithms, we propose techniques tailored to specific stages in the pipeline. Using these techniques, G^3SA achieves significant end-to-end speedup against the CPU and GPU baselines.

*Both authors contributed equally to this research.

This work is licensed under a Creative Commons Attribution 4.0 International License.

ICS '25, Salt Lake City, UT, USA
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1537-2/25/06
https://doi.org/10.1145/3721145.3729516

CCS Concepts

• **Applied computing** → **Computational genomics**; • **Computing methodologies** → **Massively parallel algorithms**.

Keywords

Sequence Alignment, Next-Generation Sequencing, Graphics Processing Unit (GPU)

ACM Reference Format:
Yeejoo Han, Sunwoo Kim, Seongyeon Park, and Jinho Lee. 2025. G^3SA : A GPU-Accelerated Gold Standard Genomics Library for End-to-End Sequence Alignment. In *2025 International Conference on Supercomputing (ICS '25), June 08–11, 2025, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3721145.3729516

1 Introduction

Since its birth in the 1970s [83], DNA sequencing has matured significantly to serve as an essential research tool and drive improvement in many fields such as genomics [82], medicine [6], and agriculture [88]. This was made possible by advances in DNA sequencers. These machines can extract large numbers of sequences with various lengths, such as short (50-300bp) and long (1k-100kbp) reads. [25] As the quality of sequencer output improved and costs decreased, the readily available large sequence data made manual analysis impractical, necessitating a pipeline that could effectively and precisely inspect the said data. This naturally led to the introduction of a wide range of DNA sequence analysis pipelines designed to provide detailed analysis depending on the needs of the user, such as variant calling [66, 99], metagenomics analysis [84], and genome assembly [13, 86].

The primary step of such DNA sequence analysis pipelines is *sequence alignment*. Given a small number of long reference sequences and a large number of (relatively) short query sequences, the goal of sequence alignment is to find the start

and end position in the reference sequence that has the highest similarity for each query sequence. Several sequence alignment libraries have been developed to improve the efficiency and accuracy of this process. [8, 34, 38, 39, 41, 44]

Among these libraries, BWA-MEM [39] and Minimap2 [41] are drawing much interest and sometimes referred to as the de facto gold standards for the analysis of short and long sequences [9, 33, 52], respectively. The two libraries have been used in milestone genomic projects: Minimap2 was used to build the first human pangenome reference [45] and to analyze the complete human reference genome [1], and BWA-MEM has been included in the pipeline recommended by the Broad Institute [90]. Additionally, various works in both academia [18, 23, 36, 37, 65] and industry [61] have explicitly targeted these libraries, highlighting their significance. Despite designed for different sequence lengths, both libraries follow a similar overall process based on the *seed-chain-extend* paradigm, as detailed in Section 2. The seeding stage extracts relatively short exact matching subsequences between the reference sequence and the query sequence. This process helps identify regions where alignment is more likely to occur. Next, the chaining stage groups seeds that are closely located in the reference sequence, further filtering the candidate regions. Finally, the extending stage computes the exact start/end position and the similarity score of the alignment between the reference and query sequences.

Despite continuous efforts such as restructuring [91], using SIMD [36] or multiple nodes [37], these alignment algorithms are failing to keep up with the rapid growth of the sequencers. For example, alignment using BWA-MEM2 can take approximately 869 hours to finish [5]. Therefore, alignment step remains one of the major bottlenecks of bioinformatics pipelines [5]. One direction of addressing this bottleneck is the use of graphics processing units (GPUs) to accelerate alignment algorithms [2, 18, 20, 62, 63, 65]. As GPUs are becoming a widespread choice of accelerators for graphics [81, 96], machine learning [16, 68], or data analytics [54, 64], GPU-accelerated alignments could be promising solutions to dramatically improve the processing speed.

Given a large body of effort on GPU-based accelerations [3, 17, 19, 46, 49, 51, 53, 62, 63, 78, 79, 95], one possible approach would be to collect the necessary pieces to put together an end-to-end GPU-executable replacement. Unfortunately, our attempt reveals that there are still missing pieces, and naively implementing and combining those yields a substantial amount of execution time overhead. For example, there are optimizations to some stages that significantly impact the computation results, and some auxiliary stages have not been implemented yet into GPU kernels. Although sequence alignment can be performed with reasonable accuracies without those auxiliary stages, providing them as is would have a meaningful value, as practitioners prefer known algorithms because modifying the sequence alignment algorithm can affect the entire genomics pipeline [31, 94], and assessing this impact requires extensive field analysis.

To bridge the gap and foster advanced research, we present G^3SA (GPU-accelerated Gold standard Genomic Sequence Alignment), the first end-to-end GPU-accelerated sequence alignment framework of the gold standard libraries, BWA-MEM and Minimap2 for short and long reads, respectively. This is achieved by first identifying the main bottlenecks of each stage in the seed-chain-extend paradigm and providing novel optimization techniques tailored to the GPU while keeping the original algorithm intact.

For the seeding stage, we propose a *unidirectional double elongation* strategy for acceleration. Next, for the chaining stage, we devise a parallelization technique named *tiled score generation* that is tailored for GPUs to enhance memory locality. Finally, for the extending stage, we address the traceback part that was often overlooked in many previous approaches with *gridded traceback*. G^3SA outperforms the CPU baselines by at most 6.21× and the naive end-to-end GPU baseline by at most 2.07×. Additionally, we achieved at most 9.73× speedup when comparing each segment independently, as described in Section 5.

In summary, our contribution is the following:

- To the best of our knowledge, G^3SA is the first full end-to-end GPU acceleration framework for gold standard sequence alignment libraries for short and long sequences.
- G^3SA provides tailored schemes for each stage of the widely used seed-chain-extend paradigm.
- We devise unidirectional double elongation, tiled score generation, and gridded traceback for seeding, chaining, and extending stages, respectively.
- We evaluated our scheme across various datasets and achieved at most 1.80× and 6.21× end-to-end speedup over the CPU baseline using one and four GPUs, respectively.

2 Background

2.1 The Seed-Chain-Extend Paradigm of Sequence Alignment

Sequence alignment algorithms implement the seed-chain-extend paradigm, which has proven effective for approximate string matching [77].

Its procedure is illustrated in Fig. 1. It is based on the observation that a *read* (output of sequencers) corresponds to a slice of the reference genome. Although the read and the slice may vary to a limited degree due to natural variations and machine errors, it is highly likely that the reads have a substantial amount of exact-matching subsequences with

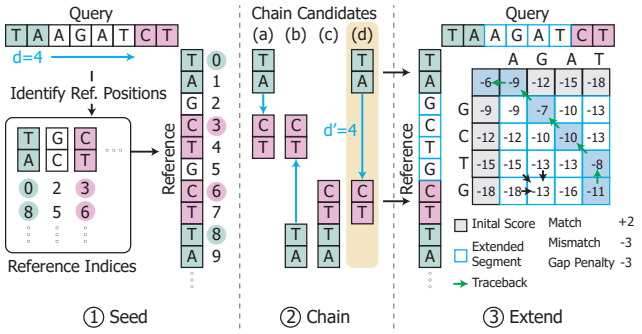


Figure 1: Overview of sequence alignment in seed-chain-extend strategy.

the reference genome. Thus, the seed-chain-extend method first collects such exact matches, namely *seeds*, based on text matching algorithms [40, 42, 67, 70]. Then, in the following *chaining* stage, those seeds are further evaluated, mainly by their position to make candidate chains. Because seeds in the final alignment appear in similar orders and positions on both the reference and target genomes, chaining gathers several seeds to provide more likely inputs to the next stage. Finally, in the *extending* stage, an approximate text matching [58, 85] is performed to calculate the scores between the seeds using dynamic programming. When the inter-seed scores are gathered to ultimately evaluate chains, the position of each read in the genome is identified, which allows building the entire genome when provided with enough reads.

Fig. 1 illustrates a toy example for the procedure of such seed-chain-extend method. In the example, a query read Q is aligned against a reference sequence R. ① For the seeding stage, a reference index stores locations of certain strings often in the form of tree [39] or hashtable [41]. In the example, matches for seeds "TA" (green) and "CT" (red) are found from the index, which corresponds to multiple locations in the reference. ② In the chaining stage, some chains—combinations of a few seeds—are examined to check feasibility. In the first chain denoted (a), the two seeds are too close to each other compared to their distance within Q. (b) has a reversed order of the seeds, and those of (c) are both reversed and too close. On the other hand, (d) has adequate orientation and a similar distance compared to Q. Thus, it is marked feasible and is passed to the extending stage. In practice, a chain consists of a few tens to thousands of seeds. ③ In the extending stage, the strings between the seeds are compared using an approximate string-matching algorithm, often using dynamic programming [58, 85].

The extending algorithm can be described as filling a two-dimensional table, where each cell at (i, j) represents a best-matching score up to $R[i]$ and $Q[j]$. In the forward phase, the cells are filled with scores. Each cell $H(i, j)$ is calculated as a variant of the below:

$$H(i,j) = max \begin{cases} 0 \\ E(i,j) \\ F(i,j) \\ H(i-1,j-1) + S(i,j) \end{cases}, \quad (1)$$

$$E(i,j) = max \begin{cases} H(i,j-1) - \alpha \\ E(i,j-1) - \beta \end{cases}, \quad (2)$$

$$F(i,j) = max \begin{cases} H(i-1,j) - \alpha \\ F(i-1,j) - \beta \end{cases}, \quad (3)$$

where terms inside the $max()$ of Eq. (1) correspond to insertion ($E(i, j)$), deletion ($F(i, j)$), and match/mismatch ($H(i-1, j-1)$). The score function $S()$ returns a positive value on a match and returns negative otherwise. The separate E and F terms exist to assign different penalties for new gaps (α) and continued gaps (β). In our example, we use gap penalties $\alpha = \beta = -3$, and +2/-3 for match/mismatch in $S()$. In practice, slightly more complicated equations are used to reflect different lengths of insertion/deletion gaps [27]. This seemingly-serial extending algorithm is known to exhibit much parallelism, especially along the anti-diagonal groups whose sum of row/col indices ($i + j$) are the same. Such parallelism is exploited in many designs with CPUs [91], dedicated accelerators [23], and GPUs [7, 62, 63, 65].

After filling the cells, the traceback phase finds out the path of the maximum-score match between the two strings. This path is represented using CIGAR strings [28], a widely adopted format to denote matches, mismatches, insertions, and deletions between two aligned sequences. To do this, the sources of each $max()$ are often stored in the memory when filling the cells as a separate ternary direction value per cell ($up, left, right$). Finally, the output is used to determine which position in R should Q be aligned to.

2.2 De Facto Gold Standard: BWA-MEM and Minimap2

BWA-MEM [39] and Minimap2 [41] are often regarded as the gold standard alignment libraries for many bioinformatics pipelines [52], each targeting short and long reads, respectively. Fig. 2 shows a simplified execution flow of the two alignment libraries. Although both employ the seed-chain-extend strategy described in Section 2.1, their detailed strategy slightly differs, mainly from the different input traits. Due to space limits, we provide a high-level overview for each stage. For a detailed description, please refer to the original papers [39, 41].

The algorithms begin with the seeding stage. BWA-MEM focuses on identifying continuous exact-matching regions,

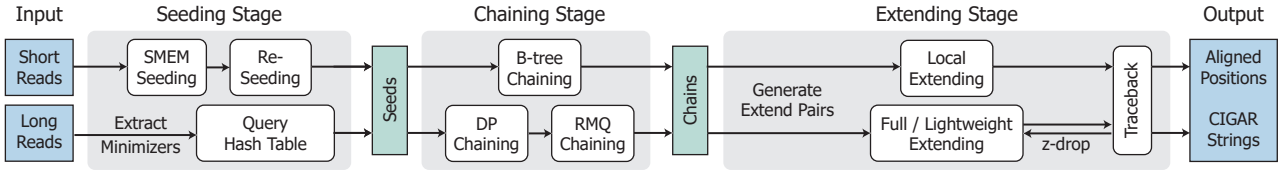


Figure 2: Overview of the workflow of gold standard sequence alignment.

leveraging the lower error rates of short reads [26]. Specifically, BWA-MEM identifies super maximal exact matches (SMEMs), the longest exact matches starting from a specific position in the query. SMEMs are constructed by iteratively elongating the seeds by looking up the adjacent query base pairs on the reference sequence, using an index tree represented by the FM-index [21]. This continues until a mismatch is encountered. To mitigate potential misalignments caused by excessively long seeds, BWA-MEM employs a re-seeding optimization that splits such seeds and performs additional seeding steps, resulting in a larger set of seeds [39].

Minimap2, on the other hand, fixes the seed length instead of generating exact seeds of varying lengths, given the longer inputs. Thus, Minimap2 begins the seeding stage by extracting minimizers [71], which are sampled subsequences of a fixed size in the query string. These minimizers serve as keys in the reference hash table, enabling fast extraction of seeds.

The seeding stage is followed by the chaining stage. BWA-MEM uses a b-tree structure to chain seeds that share a similar relative order along the query and reference. In contrast, Minimap2 uses a more complicated chaining strategy to identify the optimal set of seeds (also called anchors). It employs a chain scoring function to evaluate the relative order of anchor pairs based on their reference and query positions. Then Minimap2 uses dynamic programming (DP) to identify the chain with the highest score, ensuring high-quality alignment. The latest versions of Minimap2 introduced an additional chaining step that leverages the range minimum query (RMQ) strategy. This approach uses a simplified linear scoring function and a longer gap threshold, enhancing its performance for certain scenarios [43].

For extending, both BWA-MEM and Minimap2 generate extending-target pairs and use dynamic programming-based algorithms [85]. BWA-MEM extracts the regions before/after each seed and performs extending using BWA-MEM-specific optimization techniques (e.g., x-drop [39]). Minimap2 performs extending on the entire chain, employing a combination of full and lightweight extending for higher performance. At each chain end, full extending calculates the exact position with the maximum DP score, which serves as the starting point for traceback. Within the chain, lightweight extending is applied to efficiently identify approximate maximum scores, filling the gaps between seeds. Additionally, Minimap2 monitors the trend of the extending score along the sequence with the z-drop policy. This discards extendings that show a sudden score drop while filling the score table, ensuring more reliable alignment results. Finally, both libraries perform traceback to reconstruct the alignment path and generate the corresponding CIGAR string, which encodes the alignment details.

Table 1: Analysis on Existing GPU-based Alignment Acceleration Methods

Type	Name	Seeding		Chaining	Extending	
		Seed	Reseed	B-tree	Local	TB
Short	BWA-MEM-GPU [65]	✓	✗	✓	△	✓
	GPUseed [2]	✓	✗	✗	✗	✗
	GASAL2 [3]	✗	✗	✗	△	✓
	G^3SA	✓	✓	✓	✓	✓

Type	Name	Seeding	Chaining		Extending		
		Seed	DP	RMQ	Full	LW	TB
Long	mm2-gb [18]	✗	✓	✗	✗	✗	✗
	AGAThA [62]	✗	✗	✗	✓	✗	✗
	Manymap [20]	✗	✗	✗	✗	✓	✓
	AnySeq/GPU [56]	✗	✗	✗	△	✗	✓
	G^3SA	✓	✓	✓	✓	✓	✓

3 Challenges

3.1 Difficulty of GPU Acceleration on Gold Standards

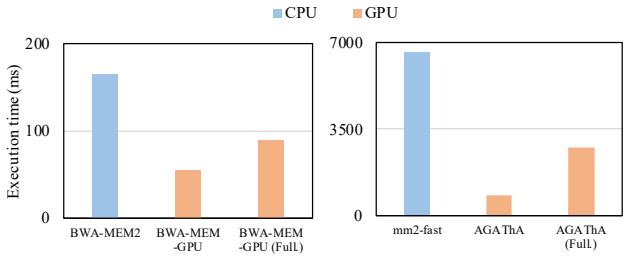
We survey existing GPU-based sequence alignment acceleration works [3, 17, 19, 46, 49, 51, 53, 56, 62, 63, 65, 78, 79, 95] and compare representative works in Table 1. Unfortunately, complete direct replacements for the gold standard libraries are yet to be developed. For example, various works [46, 49, 51, 53] suggest new algorithms to support GPU-friendly designs. While designing a new GPU-friendly algorithm is certainly a valuable direction, this does not align with our goal of providing a direct replacement of BWA-MEM and Minimap2. Additionally, prior works shown in Table 1 tend

to focus on a single workflow stage [2, 3, 18, 20, 56, 62], usually choosing from seeding, chaining, or extending. [65] is designed to perform end-to-end alignment, but scopes out the auxiliary stages such as re-seeding or banding during local extending, focusing on the core kernels. Such an implementation serves as a valid sequence alignment and could be thought of as a trade-off between alignment quality and speed, since those auxiliary stages add non-negligible overhead. However, this does not align with the goal of being direct replacements. For instance, in the seeding stage, bypassing the re-seeding step in short-read alignment will skip processing up to 70% of the total seeds in subsequent stages. Another example is the extending stage, where relying solely on the full or lightweight (LW) extending can affect the outputted aligned position. We further evaluate these trade-offs in Section 5.5.

However, implementing all missing functionalities is challenging and it is easy to result in significant performance loss. This is because the gold standard algorithms contain many control-intensive optimizations aimed at reducing the amount of computation. For example, both BWA-MEM and Minimap2 employ dropoff techniques (Section 2.2), which track local maximum scores and terminate the calculation under certain conditions. However, such a technique harms parallelism, incurs extra max-reduction overhead, and creates severe warp divergence [62].

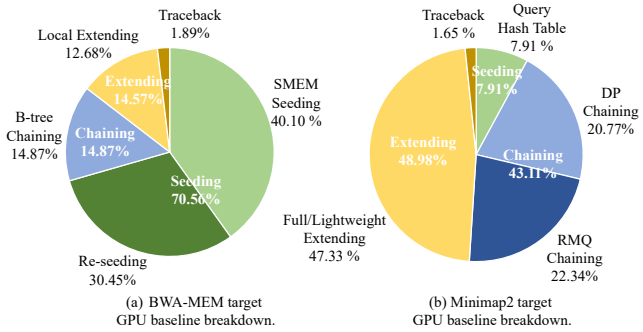
In Fig. 3, we demonstrate the challenge of implementing the two algorithms with full details for replacement. We examine the two representative stages, seeding and extending, on existing solutions BWA-MEM-GPU [65] and AGAThA [36], respectively. We modified them to fully include the auxiliary stages. The results consistently reveal that using an extending of prior works to implement direct replacements would lead to a considerable slowdown. On the seeding stage, the original BWA-MEM-GPU [65] outperforms BWA-MEM2 (CPU). However, when the seeding stage is fully implemented with re-seeding and additional heuristics, the overhead severely slows the execution by 1.8×. This strengthens the case for accelerating the seeding stage, preferably with a strategy that can adapt to both the SMEM seeding and the re-seeding computations.

A similar trend arises in the extending stage. AGAThA [62], regarded as a state-of-the-art solution for long-read extending, employs advanced techniques such as z-drop and banding [41]. However, it does not include the traceback step, requiring modification that leads to performance degradation. Notably, AGAThA's optimizations mainly focus on minimizing memory access by avoiding the storage of cell scores, which are needed for traceback. As shown in Fig. 3, AGAThA extended with traceback results in a diminished speedup compared to the original version, thereby supporting the previous claims.



(a) Seeding runtime comparison. (b) Extending runtime comparison.

Figure 3: Execution time comparison of the original algorithms and existing GPU-alignments.



(a) BWA-MEM target GPU baseline breakdown. (b) Minimap2 target GPU baseline breakdown.

Figure 4: Execution time breakdown of the implemented baselines.

These cases highlight the challenges in accelerating the end-to-end algorithms for direct replacements of gold standard algorithms on GPUs. Despite the large body of prior works, naive attempts to integrate existing solutions often lead to significant slowdowns. Based on these insights, we propose novel acceleration schemes to address these limitations effectively.

3.2 Execution Time Breakdown of the Naive Baselines

As a starting point, we first implemented both BWA-MEM and Minimap2 with existing libraries and filled in the non-existing parts with our kernels written from scratch to be our baselines. For BWA-MEM, we started from [65], and modified the seeding and extending stages to match the original algorithm. For Minimap2, we wrote seeding and chaining kernels from scratch and used AGAThA [62] for the extending kernel. Because AGAThA does not implement traceback, we modified AGAThA to create a directional matrix during the forward phase. In addition, we integrated both the full and lightweight extending into the workflow to match the algorithm of Minimap2 [41].

Fig. 4 plots the breakdown of the baselines we implemented. Similar to the insights from Section 2, extending is

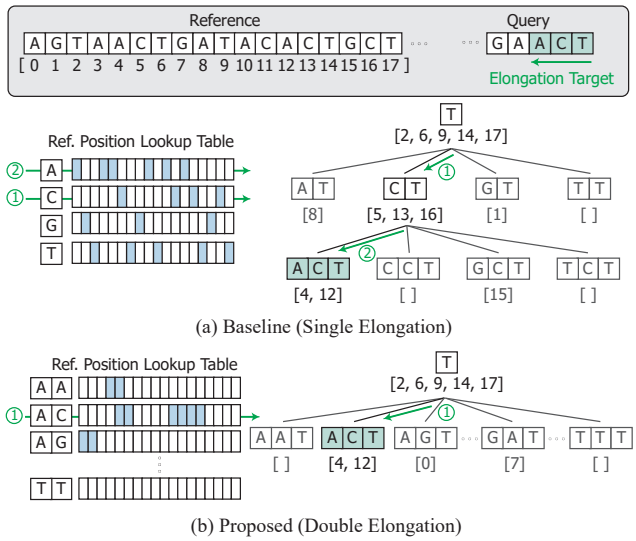


Figure 5: Design of BWA-MEM's single elongation seeding and G^3SA's double elongation seeding.

the major stage in both algorithms. The BWA-MEM baseline spends a lot of time (70.6%) on its seeding compared to the chaining, and the Minimap2 baseline spends the majority of the time on chaining (43.11%). Based on those, we will focus on how G^3SA improves the performance of BWA-MEM seeding, Minimap2 chaining, and extending for both algorithms, while also implementing all the other stages.

4 G^3SA Design

As shown in Section 2, a typical read alignment process consists of three distinct steps, making it natural to adopt a stage-by-stage approach in designing the framework. However, as demonstrated in Section 3, execution flows for short and long-read alignment exhibit different tendencies, with short-read alignment bottlenecked by seeding and long-read by chaining. Thus, for the seeding and chaining stage, we tackle the most time-consuming part of each pipeline and explain the necessary details in each section. For the shared extending stage, we propose a general scheme that could be used in both short and long-read alignment pipelines.

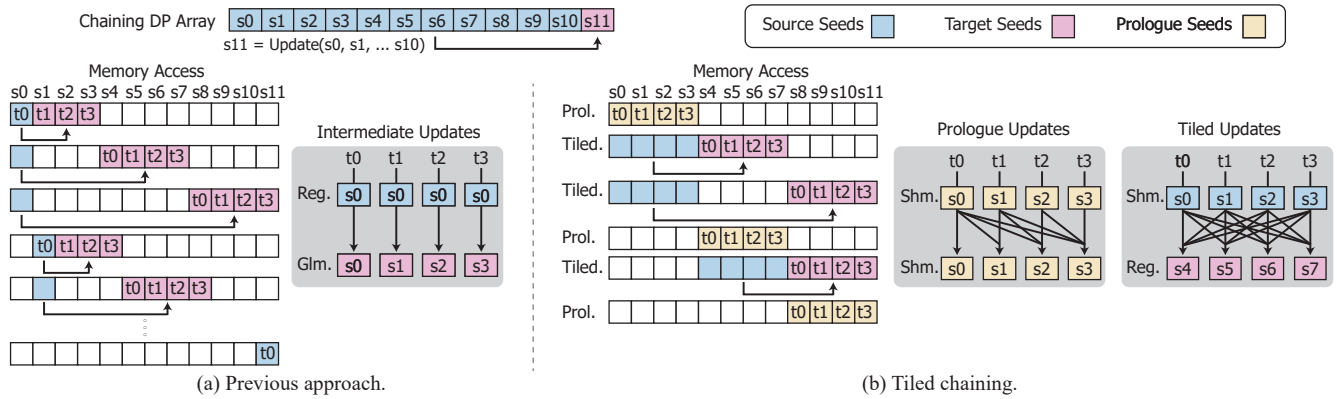
4.1 Seeding

The short-read aligner uses an iterative approach to elongate a shorter pre-seed in steps until it becomes a valid seed. This elongation requires a tree-like index structure [42] of the reference for fast operation, which should also be compact enough to be applied to the giga-base-long reference sequences. Hence the tree traversal fundamentally exhibits a random memory access pattern, easily becoming a bottleneck. CPU-based algorithms [39, 87] reduce the number of memory accesses by elongating multiple growing seeds concurrently in both directions and aggressively filtering out seeds that are subsets of other seeds. While such methods effectively reduce the number of tree lookups, they are unfriendly to GPU parallelization and would greatly suffer from severe warp divergence due to unpredictable sequence patterns.

Instead, we adopt a simple yet powerful strategy of unidirectional double-step elongation. In the original FM-index-based elongation, every single base (one of A, C, G, and T) is used as the node of the lookup tree structure as shown in Fig. 5 (a). For example, starting from base T in the query, it looks up for the backward bases C and A in two steps from the reference position lookup table in the FM-index structure [21]. Starting from T which appears in positions 2, 6, 9, 14, and 17 of the reference sequence, ① following the children for C returns the appearances of CT in positions 5, 13, and 16. ② Following the children again for A further returns the positions of ACT in 4 and 12. In the original algorithm, the elongation is additionally performed in the other (forward) direction to maximize the match length, which requires reverse-traversing of the tree.

The key idea of double-step elongation is to elongate two bases at each step at any point in the seeding stage. In effect, each tree node contains 16 children instead of four, as illustrated in Fig. 5 (b). Starting from the same T, ① it searches directly for children AC to find ACT in a single step. This approach slightly increases the amount of computation and requires handling the potential single base pair at the end, but it essentially halves the number of random memory accesses. A similar idea was previously suggested by [11], but the original work only evaluated their method to the bare exact matching with <120bp sequences. To the best of our knowledge, it has not been studied in the context of specific seeding algorithms or GPU-based implementations. Because FM-index [21] is used to represent the tree, implementing this consists of storing two base pairs instead of one for each row of the index.

In addition, we confine the threads to elongate seeds in the backward direction only. Since FM-index naturally supports elongating a seed in a backward direction only, BWA-MEM modifies the index to support forward elongation as well. BWA-MEM first converts the seed to a reverse complement form, performs a backward elongation with the complement, and then converts the elongated seed back to the original form [39]. Although this adds computation overhead and extra state information per elongated seed, previous works [2, 65] followed this configuration to elongate seeds in a forward direction. This overhead becomes more significant as the index is modified to support elongating multiple bases, potentially negating the performance benefit. Hence,



(a) Previous approach.

(b) Tiled chaining.

Figure 6: Minimap2 Chaining Optimization.

we have each thread elongate in the backward direction only, eliminating redundant computation and space overhead.

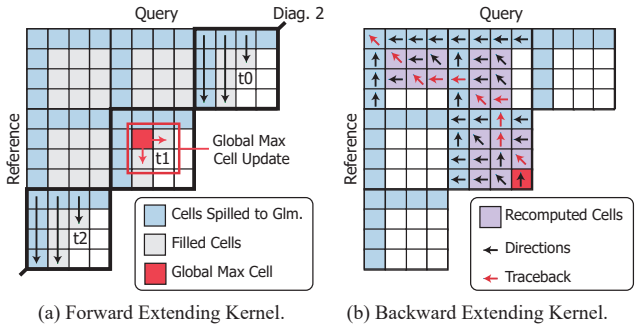
4.2 Chaining

The original chaining algorithm of Minimap2 is serial and is not suitable for parallel processing on GPU, because of the frequent reduction of the seeds. Some approaches on CPU [29, 75] try to address this by switching the computational order and replacing the reduction operation with parallel updates. Another issue with chaining is the irregular length of chain ranges, spanning from a few to several thousand. This results in workload imbalance and underutilization of GPU cores. The state-of-the-art chaining kernel, mm2-gb [18] addresses this issue by read segmentation and range bucketing. However, as mentioned in the same paper, the chaining stage is still bottlenecked by chains with longer ranges ($> 1k$) that take much longer for score generation. To amend this, we propose a novel parallelization scheme named *tiled score generation* for chaining, targeting long chains by improving the temporal locality of thread-local registers and shared memory.

Fig. 6 illustrates the procedure of the chaining stage. From an array of seeds, the goal is to update the scores for all cells that individually correspond to the seeds. In the chaining, scores are calculated from all its predecessors by function $Update()$, as denoted by the equation in the top-left of the figure. The $Update()$ function is commutative, and previous approaches shown in Fig. 6 (a) typically exploit this characteristic. They take one predecessor cell as a source, update the target scores of the succeeding cells in parallel, and then move to the next cell as a source predecessor. However, as the chaining range gets longer, we cannot fit all the cells into the shared memory, which leads to excessive redundant global memory access. For example, in Fig. 6 (a), the cell at the far end of the score array (s11) is overwritten 11 times throughout the entire process, and each access is spread across iterations.

In contrast, Fig. 6 (b) illustrates the proposed *tiled chaining* approach that exploits the locality of both the source and target cell accesses. Instead of taking a single source for all the threads, multiple cells are taken as sources, and the same number of cells are taken as targets. With this, both source and target can reside in the shared memory with fewer iterations. Because of the commutativity, the functionality is preserved.

The tiled chaining operates in two steps: prologue and tail update. The prologue step is taken when the source and the target seeds are identical. The cells are loaded to the shared memory, and updates occur in a shuffling pattern from lower threads to higher threads according to the corresponding seed information. The tail update step is taken when the source cells and the target cells are different. The source cells are still in the shared memory, as it was already loaded in the previous prologue step, and we load the target cells in the registers. As shown in the figure, all the source seeds have smaller indices than the target seeds. Because of this, the updates occur in an all-to-all pattern. In the pattern, all the reads are from the shared memory, and the writes are private to the register, which can be done efficiently without any conflicts. The kernel repeats the tiled update until it reaches the end of the range. This allows us to reduce unnecessary global memory access by the tile size. Our approach is particularly beneficial for long chains, where the entire anchor array cannot fit into the L1 cache space and is thus bottlenecked by excessive global memory access. After the aforementioned DP chaining is complete, the anchors selected by their long length are passed through an additional *RMQ chaining* stage. This RMQ step uses a simpler linear score and a tree-like structure to reduce the computational burden of the CPU for large anchors. However, using an additional tree structure can significantly slow down the process if used in the GPU



(a) Forward Extending Kernel. (b) Backward Extending Kernel.

Figure 7: Forward extending algorithm and gridded traceback.

naively. Therefore, we modify the RMQ chaining part to use the same seed array structure as DP chaining, and only modify the score function to match the original RMQ chaining results. This allows us to not only output the same chaining results of Minimap2 but also maintain a high performance.

4.3 Extending

As described in Section 2, the extending stage is composed of the forward phase and traceback phase. There are several prior works that target the extending stage, but they target a single type of forward phase [62, 63, 65], and often omit the traceback part [62, 63].

4.3.1 Forward. The forward phase of the extending stage calculates the similarity score between the reference and query. It uses two patterns, named full extending and lightweight extending[1].

Full extending. Full extending refers to the widely used approximate string-matching algorithms such as Smith-Waterman (local alignment) and Needleman-Wunsch (global alignment). As shown in Fig. 7 (a), the threads parallelly fill blocks (4×4 cells in the figure) in anti-diagonal positions (thick boundaries), which is a common strategy to exploit parallelism from Eq. (1)-Eq. (3). It is named full extending because it searches for the exact global maximum value among the score table. Accelerating these algorithms has been vigorously studied in several prior works [3, 62, 63].

Lightweight extending. To achieve better performance with nearly identical results, Minimap2 introduces an optimization method called lightweight extending. Instead of calculating the exact global maximum value, it computes the approximate maximum value by greedily maintaining the global maximum cell. The threads collectively maintain a single global max cell, initialized as the upper-leftmost cell. When the cells around the current global maximum are calculated, the maximum-valued neighbor is selected as the next global max (depicted in red).

We design our lightweight extending kernel based on a state-of-the-art exact extending kernel, AGAThA [62]. Unlike the full extending kernel, we do not need to keep track of all the anti-diagonal max scores, relieving the prior design's shared memory constraint. Thus, we optimize the lightweight extending kernel by carefully placing the boundary scores onto the shared memory, minimizing global memory spill.

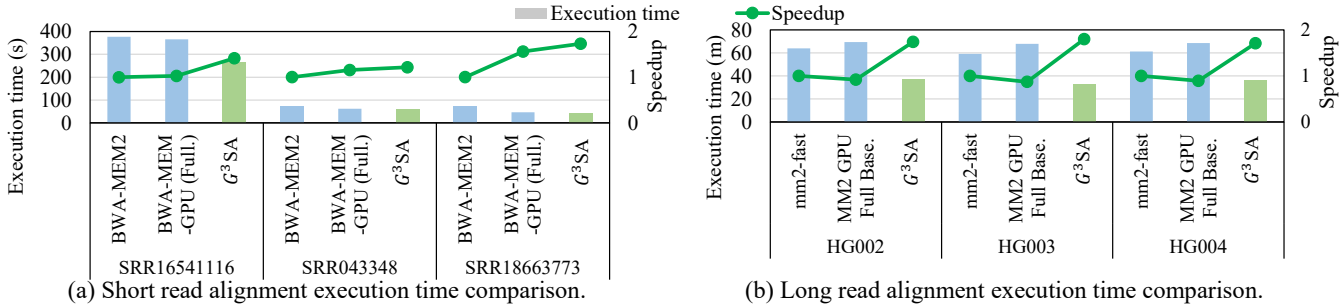
4.3.2 Traceback. Once the forward phase score computation is done, we start from the global maximum score cell, tracing back the optimal alignment path. To perform traceback, a typical approach is to store a pre-calculated direction matrix during the forward phase. Each cell of the direction matrix contains information on where the corresponding score of the current cell originates from (up, left, or diagonal) Eq. (1).

This contradicts many existing methods optimized for the forward phase. For example, a commonly used technique is to place the cell scores in GPU shared memory to reduce the global memory traffic [7, 62, 63]. While this approach is promising for the forward phase, it makes traceback infeasible, as the direction matrix has not been saved. As illustrated in Fig. 1, traceback requires the individual directions matrix passed from the forward phase to determine the trace direction. One way is to dump the direction matrix from shared memory to global memory, but this defeats the purpose of using shared memory for memory efficiency. When naively implemented, such a method will significantly slow down the forward phase execution.

To address this issue, we propose a novel scheme called gridded traceback as shown in Fig. 7 (b). From the figure, it is evident that the values in the matrix far from the traceback path (red arrows) do not contribute to the traceback computation. To take advantage of this, we store partial score data in the global memory and recalculate only the cells needed for traceback during the traceback.

During the forward phase, the scores are first stored in the shared memory as in prior works [62, 63, 65]. Then, a few scores at the boundary of the predefined grids (blue) are written to the global memory. When the traceback starts from the global max cell, the nearest upper-left grid boundary cells are loaded, and only the necessary scores (purple cells) are recalculated as in the forward phase. After the traceback is performed within the block, the recalculated scores in shared memory are deleted as they are no longer needed. Then, the same recalculation-based traceback is performed on the next target grid until the trace reaches the starting point.

[1]These are called 'exact' and 'approximate' in the original Minimap2 library, but we use the term 'lightweight' instead to avoid confusion with exact and approximate end-to-end algorithms of sequence alignment.



(a) Short read alignment execution time comparison.

(b) Long read alignment execution time comparison.

Figure 8: End-to-end performance evaluation of G^3SA with short- and long-read datasets against respective CPU and GPU baselines.

Gridded traceback essentially provides a trade-off between memory bandwidth and computation. However, provided that the extending is memory-bound and computation is cheap in GPUs, the benefit of memory access reduction is more significant than the additional computation. By strategically managing this trade-off and selecting appropriate grid sizes (as detailed in Section 5.4), we could achieve both reduced latency and lower memory usage for the combined extending stage. The advantages of gridded traceback escalate as the target sequences become longer. This is because the traceback matrix size increases quadratically with sequence length, whereas the number of recomputed grids tends to increase linearly.

4.4 Additional Details

In addition to the three proposed techniques, we faithfully implemented the sub-stages of the gold standard algorithms depicted in Fig. 2. In the short read, we included the additional re-seeding after the aforementioned seeding kernel, which improves alignment quality [39]. We also implemented the original optimization techniques for extending such as early terminations and banding. In the long read, we have implemented the workflow of Minimap2 to seamlessly connect each stage. Essential post-processing steps, such as chain backtracking and identifying primary chains, were ported and parallelized. To fill in the gap between the seeds within a chain and form a complete alignment, we identify the segment pairs from each chain, collect them, and perform extending in batches. We handle a few exceptional cases of reads with complex branching by re-sending them to the CPU, which could overlap with other kernel operations on the GPU. All kernels have been optimized with the best of our effort with parallelization, optimizing memory systems, and fusing kernels when possible. Because we omit the miscellaneous details in the paper for space reasons, please refer to the released code for individual implementations.

5 Evaluation

5.1 Experiment Setup

We evaluated G^3SA against various baselines on diverse datasets. As baselines, we select the state-of-the-art CPU and GPU implementations of the gold standard read aligners-BWA-MEM for short reads and Minimap2 for long reads.

For short-read alignment, we use the following baselines:

- BWA-MEM2 [91] is the CPU-accelerated version of BWA-MEM. We use this as the reference for gold standard short-read alignment.
- BWA-MEM-GPU [65] is a GPU acceleration for the core kernels of BWA-MEM.

For long-read alignment, we use the following baselines:

- mm2-fast [36] is a SIMD-accelerated library of Minimap2, regarded as the current gold standard.
- mm2-gb [18] is the state-of-the-art implementation for the GPU chaining stage of Minimap2, whose subsequent stages are executed on the CPU. We use this work to evaluate our chaining performance.
- AGAThA [62] is a state-of-the-art forward extending kernel that produces nearly exact results to Minimap2. Since this work only implements the forward extending kernel, we modified the code to perform traceback.
- Manymap [20] is a Minimap2-based extending kernel with both the forward and traceback kernel. It only provides lightweight extending, but we include this as a baseline with traceback support.

Since there is currently no end-to-end GPU implementation of Minimap2, we use a naively implemented GPU version of Minimap2 mentioned in Section 3.2. For stage-wise GPU performance comparison, we compare our DP chaining kernel with mm2-gb, and our extending kernel with Manymap and AGAThA.

We used real-world human DNA datasets for both short and long reads. For reference genome, we used the human genome GRCh38 [69] and the E. coli genome [24]. This

reference genome was preprocessed to build indices for both short and long reads using the gold standard CPU libraries [39, 41]. For short reads, we utilized Illumina sequences of varying lengths from the NCBI database [57]. Specifically, SRR16541116, SRR043348, and SRR18663773 were used, with read lengths of 75-148 bp. For long reads, we used HG002, HG003, HG004 PacBio reads from genome-in-a-bottle project [59]. 7 million reads from each dataset were used for evaluation. For stage-wise evaluation, the intermediate data from the previous stage was pre-generated and provided as input. All experiments were run on a workstation with an AMD Ryzen Threadripper PRO 7985WX CPU (128 hardware threads) and NVIDIA RTX A6000 GPUs. We use a single A6000 GPU for experiments unless noted otherwise.

5.2 Performance Comparison

In this section, we evaluated G^3SA's performance on the two gold standard aligners, BWA-MEM and Minimap2. We compared the end-to-end execution time against the CPU-based implementations and their respective GPU-based baselines we described in Section 3.2.

For short reads, we compared the execution time and speedup of G^3SA against BWA-MEM2 and BWA-MEM-GPU in Fig. 8 (a).process, even when the data comprises short sequences. With all the necessary kernels added, BWA-MEM-GPU achieves 1.23× geomean speedup. However, this baseline does not fully utilize the GPU due to the memory overheads in seeding and extending. G^3SA, in contrast, achieves 1.44× geomean speedup by optimizing the two bottlenecks of BWA-MEM.

For long reads, Fig. 8 shows the overall execution time and speedup of G^3SA and the naive GPU baseline compared to Minimap2. Due to the extended length, reads are loaded into GPU memory in smaller batches, making it more challenging to exploit query-level parallelism than shorter reads. Additionally, the length of long read sequences exhibits significant irregularity, varying from 1kbp to around 100kbp. This contributes to workload imbalance and under-utilization. Consequently, a naive porting of the CPU version fails to leverage the parallel capabilities inherent to the GPU architecture, resulting in performance degradation, as shown in Fig. 8. Considering this issue, G^3SA targets the bottlenecks of chaining and extending, each with tiled chaining and gridded traceback, respectively. G^3SA achieved 1.95× geomean speed up over the naively ported GPU baseline and 1.75× geomean speed up over mm2-fast.

5.3 Stage-Wise Performance Comparison

To demonstrate the advantages of each proposed scheme, we compare each stage of the analysis pipeline with the existing CPU and GPU baselines.

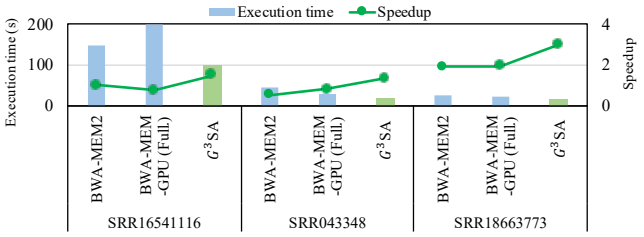


Figure 9: Evaluation of the seeding stage.

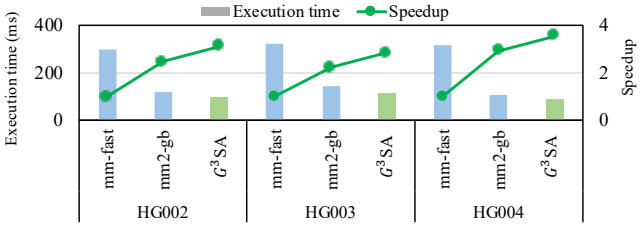


Figure 10: Evaluation of chaining stage.

5.3.1 Seeding. For seeding, we compared the execution time of BWA-MEM-GPU, and G^3SA against BWA-MEM2 in Fig. 9. Compared to BWA-MEM2, BWA-MEM-GPU shows 1.06× geomean speedup, respectively. However, both baselines require many inefficient random global memory accesses during the FM-index lookup step. In contrast, G^3SA reduces this issue using double elongation seeding, effectively reducing the number of random memory accesses by half. This leads to a geomean speedup of 1.81×, surpassing that of the baselines.

5.3.2 Chaining. Fig. 10 depicts the performance comparison of the chaining stage against mm2-fast [36] and mm2-gb [18] on GPU. mm2-gb kernel implements the single-source parallel algorithm depicted in Fig. 6 (a). Even though it achieves high speedup over mm2-fast, through range bucketing, it misses the opportunity for further speedup from utilizing the locality of the source cells. On the other hand, G^3SA achieves further speedup by handling multiple sources and target seeds at the same time, which exploits locality by placing them in the shared memory. Thus, G^3SA achieves overall 3.15× geomean speedup over mm2-fast and 1.24× geomean speedup over mm2-gb.

5.3.3 Extending. For the last stage, extending, we use mm2-fast as the baseline to compare the performance of AGAThA [62] and G^3SA in Fig. 11. Since AGAThA omits the traceback step, we modified the code to write the direction matrix packed in 4 bits. For gridded traceback, we choose a grid size of 64, carefully balancing the tradeoff between global memory writes and score recomputation. We adjusted the batch size for each setting so that the total memory usage

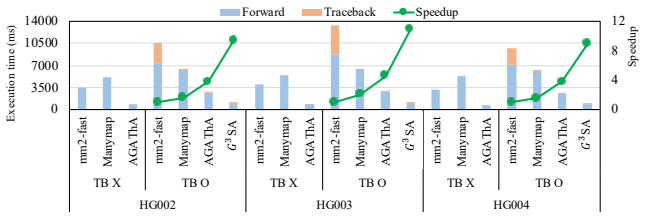


Figure 11: Evaluation of forward and traceback phase of extending stage.

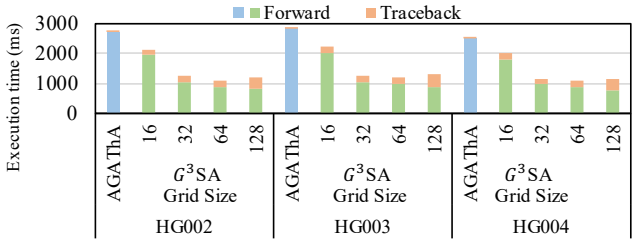


Figure 12: Sensitivity study on traceback grid size.

stays constant. Details on this design choice are elaborated in Section 4.3.2. To isolate and correctly measure the performance of the extending stage, we also include baselines that exclude the traceback step. In general, for AGAThA, the forward step execution time increased compared to AGAThA without traceback. This is because of naively writing the direction matrix during the forward process, which led to only 4.06× geomean speedup for AGAThA with traceback. On the other hand, gridded traceback achieves up to 9.73× over mm2-fast and 2.47× improvement over AGAThA. This was first done by removing the unnecessary memory access during the forward step. While the traceback time increased from additional recomputation in gridded traceback, the execution time decrease in the forward step was larger. Another important factor was batch size. As the baseline traceback requires quadratic memory space for the direction matrix, it requires smaller batch sizes, reducing the utilization of GPU streaming multiprocessors. The score grid of gridded traceback consumes less memory, enabling improved parallelism and speedup. This method has been shown to be especially beneficial when porting the end-to-end algorithm, as there is limited space available in global memory due to the presence of other intermediate data.

5.4 Sensitivity Study

For a detailed evaluation, we conducted a sensitive study for our gridded traceback extending scheme. When we change the grid size in the gridded traceback, we are trading off the memory accesses of the forward phase and the computation of the traceback phase. With larger grids, the amount of scores spilled to the global memory decreases as only the boundary values will be stored. Also, the kernel could benefit from increased parallelism with a larger batch size. However, during the traceback, the amount of recomputation of the cells to restore the directions increases because (depending on the route taken) roughly half of the cells in each grid on the traceback route are needed. To see the trade-off, we swept the grid size from 16 to 128 and compared to AGAThA [62]. The breakdown shown in Fig. 12 demonstrates the aforementioned trade-off of increasing traceback time and decreasing forward time as the size of the grid increases. All sizes within 16 to 128 are faster than AGAThA, equivalent to having 1 as the grid size. Throughout the datasets, we observe that the sweet spot is found around 64. In addition, the performance is not very sensitive around the optimal setting, making gridded traceback practical to use without intensive tuning. From the experiments, we use 64 as G^3SA's main grid size configuration for the traceback evaluation.

Table 2: Comparison of Reproducibility of Gold Standard Libraries

Stage	Acc. Frameworks	Coverage	
Seeding	BWA-MEM-GPU	29.18%	
	G^3SA	100%	
Stage	Acc. Frameworks	Accuracy	
Chaining	mm2-gb	100%	
	G^3SA	100%	
Stage	Acc. Frameworks	Accuracy	Traceback?
Extending	Manymap	97.85%	O
	AGAThA	99.68%	X
	G^3SA	100%	O

5.5 Reproducibility

Among the three major stages G^3SA has optimized, naively comparing the performance against the previous method could be inadequate because they differ in reproducibility of the target algorithms. To highlight the differences, we compared the reproducibility of the three stages for gold standard sequence alignment libraries in Table 2. We separately evaluated each stage with different metrics. For seeding, we compare the reproducibility using the 'coverage' metric, which can be defined as the ratio of overlapping seeds between the gold standard libraries and baselines. Next, for chaining, we assess the baselines using the 'accuracy' metric, which represents the percentage of seeds that are chained with the same seeds from the gold standard libraries. For extending, we first denote whether traceback is implemented

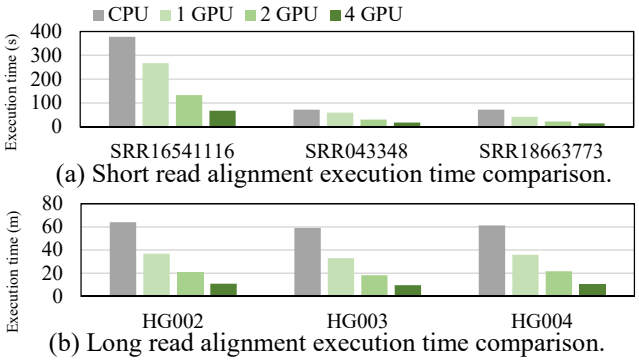


Figure 13: Scalability evaluation of G^3SA across one to four A6000 GPUs.

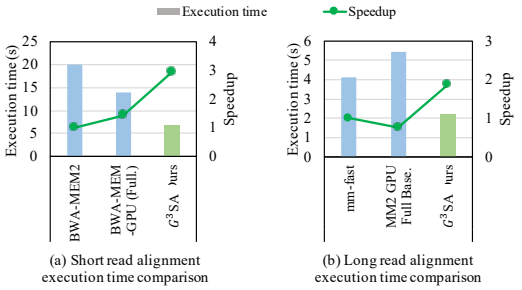


Figure 14: End-to-end speedup on E. coli dataset.

or not. Baselines that do not have the traceback algorithm fail to produce vital CIGAR string outputs from sequence alignment (e,g., AGAThA). We also compare the accuracy of the forward kernel, which is measured by the percentage of the resulting maximum score and position that correctly reproduces the results. Minimap2 utilizes distinct extending strategies within a chain, applying a full extending kernel with z-drop for the boundary seeds, while employing a lightweight extending kernel to fill the gaps between the internal seeds. However, previous works omit several details in the extending kernel design, which affects the reproducibility.

We first compared the reproducibility of short-read seeding of G^3SA with BWA-MEM-GPU and GPUSeed for short reads. G^3SA generates seeding results of high coverage of 100%. This is possible from G^3SA additionally implementing re-seeding, as detailed in Section 4.4. Next, we evaluate the long-read DP chaining kernels of mm2-gb and G^3SA. Both kernels achieve 100% accuracy, reproducing the result from Minimap2. Lastly, for extending, G^3SA reaches 100% accuracy, while Manymap and AGAThA show lower accuracy. This is due to the lack of full extending kernel and early termination strategy for Manymap, and lightweight extending for AGAThA. Note that for both short and long reads, G^3SA not only accelerates all seed-chain-extend stages but also ensures full reproducibility across all stages.

5.6 Multi-GPU Scalability

We demonstrate the scalability of G^3SA over multiple GPUs in Fig. 13. Sequence alignment typically takes large batch of independent queries as its input, and simple query-level parallelism is effective to scale its performance across multiple GPUs. We also straightforwardly duplicated the reference data and evenly distributed independent alignment tasks across the GPUs. Additionally, we noticed that reading and parsing input data from storage could bottleneck scaling G^3SA across multiple GPUs. Hence we utilized multiple host threads and concurrent queues for parallel data load to support efficient scaling of G^3SA. The results are shown in Fig. 13, which depicts the performance measured using up to four A6000 GPUs in a single node with the benchmark datasets. As shown in the chart, G^3SA achieves a near-linear speedup of 3.4× in geomean with four GPUs. This is as expected, and the slight loss of speedup is stemming from the small workload imbalance and the initial communication to send the inputs.

5.7 Extending G^3SA to a Non-Human Dataset

While human sequences are primarily targeted in genomics applications, users also process non-human data for various uses. We show the extensibility of G^3SA to sequences from other species, using 100k reads from sequencing the representative E. coli bacteria. For short reads, SRR31728524 with 150bp length was used, and for long reads, SRR31486699 was used. All datasets were obtained from NCBI database. [57] Fig. 14 shows the end-to-end performance measurement on the new dataset. G^3SA achieved 2.94x speedup for short reads and 1.82x speedup for long reads over the CPU baselines, which is consistent with the human dataset experiments. These results demonstrate the robustness of G^3SA across diverse datasets, highlighting its potential for broader applications in genomic analysis.

6 Related Works

6.1 Sequence Alignment on GPUs

There have been many families of work that target sequence alignment on GPUs. Many of those serve as stand-alone tools with novel algorithms. CUDAlign family [17, 19, 78, 79], and the CUSHAW++ family [49, 51] are a few representative examples, with some software along the similar line [47, 48, 50]. These works are mostly intended to be a new platform, with their own algorithms. In contrast, some work such as GPUSeed [2], mm2-gb [18], and Manymap [20] attempt

to implement partial stages for the gold- standard algorithms. GPUSeed was intended to implement the seeding of the BWA-MEM, mm2-gb targets the chaining stage, and Manymap targets the extending stage. However, they only implement one stage of the algorithm, making it difficult to use in the field. Furthermore, GPUSeed and Manymap are found to provide slightly altered versions compared to the original ones. Recently, there also have been attempts to utilize hybrid platforms, such as RabbitSAlign [93], which computes seeding on CPU with a novel type of seeds built on StrobeAlign [76], and computes extending stage on the GPU. BWA-MEM-GPU [65] provides an end-to-end implementation for BWA-MEM. However, it only covers a partial execution flow from the original gold standard algorithm, whose impact on the result has to be further verified. Lastly, NVIDIA's closed-source proprietary software Parabricks [61] provides implementations of both BWA-MEM and Minimap2. However, it cannot be modified or inspected due to its unpublished implementation details. In contrast, we open-source the implementation of G^3SA for the research community.

6.2 Acceleration on Various Platforms

Because sequence alignment has been studied for decades, there have been various attempts to accelerate it on CPUs. [72–74, 92] are popular implementations for Smith-Waterman algorithm, which forms a core of the extending stage, and [15] provides an optimized traceback algorithm. In addition, many libraries, including our target gold standard algorithms [30, 41, 91], have undergone several revisions for performance optimizations such as multithreading, SIMD, and/or data layouts. Several works proposed methods to realize multi-base lookups in seeding for CPUs. A concurrent work [98] utilizes a two-step FM-index similar to that of G^3SA. [87] designs a radix tree to support multi-base lookup and devise compression and reuse strategy. [35] trains a machine learning model to predict seed location. Notably, [87] and [35] require 90GB memory footprint for practical reference sequences like the human genome, requiring further optimization to run on GPUs.

Another popular direction is to create a custom dedicated accelerator for sequence alignments. For example, [97] introduced a Smith-Waterman [85] algorithm implementation on an Altera FPGA exploiting anti-diagonal parallelism. Later works followed to support banding heuristics [12, 30] or provide flexibility in input sizes [4, 32]. A popular solution in the field is Illumina DRAGEN [55]. It provides an FPGA-based tool for integrated alignment and variant calling, but its implementation details are unknown, and intermediate results cannot be obtained.

Similar to the GPU-specific software cases, many attempts have been made to co-design the hardware and software for efficiency. Darwin [89] is one of them using ASIC, with its gapped filtering greatly reduces the computational burden. Other proposals include GenAx [22] for automata-based accelerator, GemASM [10] for bit-vector-based algorithm, or SeedEx [23] banded Smith-Waterman algorithm.

While these works provide a tailored hardware solution for the target application, it is difficult to use in real-world scenarios where the majority of users in the field lack such tools. In contrast, G^3SA provides a ready-to-use solution on GPUs, which are relatively easy to obtain and use.

7 Discussion

Starting from the Hopper [14] generation, NVIDIA GPUs support a new extension to the instruction set named DPX (Dynamic Programming X). According to the documents [60] and a prior work [80], using DPX can provide several times speedup to the Smith-Waterman algorithm, which forms the core of the extending stage. This could be applied to our work as well, since the inner loop of the chaining and extending kernels consists of computations that are supported by DPX instructions. Since the proposed schemes primarily target memory access overhead, the final optimized kernels are not memory-bound. Considering this, we expect introducing the DPX will provide considerable speedup to G^3SA, orthogonal to the proposed techniques. Although Hopper GPUs are not widely available in consumer-grade or academic settings, integrating the DPX instruction offers a promising area for future exploration.

8 Conclusion

We present G^3SA, the first gold standard sequence alignment library with end-to-end GPU acceleration. G^3SA identifies similarities and bottlenecks in libraries targeting short and long sequences and accelerates each stage by unidirectional double elongation, tiled score generation, and gridded traceback for the seeding, chaining, and extending stages, respectively. It outperforms other CPU and GPU baselines with full implementations of gold standard libraries, providing genomics researchers with a new open-source tool for fast and precise sequence alignment.

Acknowledgments

This work was partially supported by National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (2022R1C1C1011307, 2020M3H2A1078119, RS-2024-00416859) and Institute of Information & communications Technology Planning & Evaluation (IITP) (RS-2024-00395134, RS-2024-00347394, RS-2023-00256081, RS-2021II211343). Jinho Lee is the corresponding author.

References

[1] Sergey Aganezov, Stephanie M Yan, Daniela C Soto, Melanie Kirsche, Samantha Zarate, Pavel Avdeyev, Dylan J Taylor, Kishwar Shafin, Alaina Shumate, Chunlin Xiao, et al. 2022. A complete reference genome improves analysis of human genetic variation. *Science* 376, 6588 (2022), eabl3533.

[2] Nauman Ahmed, Koen Bertels, and Zaid Al-Ars. 2020. Efficient GPU Acceleration for Computing Maximal Exact Matches in Long DNA Reads. In *ICBBB*.

[3] 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. *Bioinformatics* 20 (2019), 1–20.

[4] Nauman Ahmed, Vlad-Mihai Sima, Ernst Houtgast, Koen Bertels, and Zaid Al-Ars. 2015. Heterogeneous hardware/software acceleration of the BWA-MEM DNA alignment algorithm. In *ICCAD*.

[5] Mohammed Alser, Joel Lindegger, Can Firtina, Nour Almadhoun, Haiyu Mao, Gagandeep Singh, Juan Gomez-Luna, and Onur Mutlu. 2022. From molecules to genomic variations: Accelerating genome analysis via intelligent algorithms and architectures. *Computational and Structural Biotechnology Journal* 20 (2022), 4579–4599.

[6] Matthew W Anderson and Iris Schrijver. 2010. Next generation DNA sequencing and the future of genomic medicine. *Genes* 1, 1 (2010), 38–69.

[7] Muaaz G Awan, Jack Deslippe, Aydin Buluc, Oguz Selvitopi, Steven Hofmeyr, Leonid Oliker, and Katherine Yelick. 2020. ADEPT: A domain independent sequence alignment strategy for GPU architectures. *Bioinformatics* 21, 1 (2020), 1–29.

[8] Novocraft Technologies Sdn Bhd. 2023. Novoalign: High-Sensitivity Sequence Alignment Tool. https://www.novocraft.com/products/novoalign/ Accessed: 2025-04-07.

[9] BioBam Bioinformatics. 2023. Long-Read Sequence Alignment with Minimap2. https://www.biobam.com/long-read-sequence-alignment-with-minimap2/. Accessed: 2025-04-07.

[10] Damla Senol Cali, Gurpreet S Kalsi, Zülal Bingöl, Can Firtina, Lavanya Subramanian, Jeremie S Kim, Rachata Ausavarungnirun, Mohammed Alser, Juan Gomez-Luna, Amirali Boroumand, et al. 2020. GenASM: A high-performance, low-power approximate string matching acceleration framework for genome sequence analysis. In *MICRO*.

[11] Alejandro Chacón, Juan Carlos Moure, Antonio Espinosa, and Porfidio Hernández. 2013. n-step FM-index for faster pattern matching. *Procedia Computer Science* 18 (2013), 70–79.

[12] Peng Chen, Chao Wang, Xi Li, and Xuehai Zhou. 2013. Hardware acceleration for the banded Smith-Waterman algorithm with the cycled systolic array. In *FPT*.

[13] Yu Chen, Yixin Zhang, Amy Y Wang, Min Gao, and Zechen Chong. 2021. Accurate long-read de novo assembly evaluation with Inspector. *Genome Biology* 22 (2021), 1–21.

[14] Jack Choquette. 2023. NVIDIA Hopper H100 GPU: Scaling Performance. *IEEE Micro* 43 (2023), 9–17.

[15] Rezaul Alan Chowdhury, Hai-Son Le, and Vijaya Ramachandran. 2008. Cache-oblivious dynamic programming for bioinformatics. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 7, 3 (2008), 495–510.

[16] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *NeurIPS* (2022).

[17] Edans Flavius de Oliveira Sandes, Guillermo Miranda, Xavier Martorell, Eduard Ayguade, George Teodoro, and Alba Cristina Magalhaes Melo. 2016. CUDAlign 4.0: Incremental speculative traceback for exact chromosome-wide alignment in GPU clusters. *IEEE Transactions on Parallel and Distributed Systems* 27, 10 (2016), 2838–2850.

[18] Juechu Dong, Xueshen Liu, Harisankar Sadasivan, Sriranjani Sitaraman, and Satish Narayanasamy. 2024. mm2-gb: GPU Accelerated Minimap2 for Long Read DNA Mapping. *bioRxiv* (2024), 2024–03.

[19] F de O Edans, Guillermo Miranda, Alba CMA de Melo, Xavier Martorell, and Eduard Ayguadé. 2014. CUDAlign 3.0: Parallel biological sequence comparison in large GPU clusters. In *CCGrid*.

[20] Zonghao Feng, Shuang Qiu, Lipeng Wang, and Qiong Luo. 2019. Accelerating long read alignment on three processors. In *Proceedings of the 48th International Conference on Parallel Processing*.

[21] Paolo Ferragina and Giovanni Manzini. 2001. An experimental study of an opportunistic index. In *SODA*.

[22] Daichi Fujiki, Arun Subramaniyan, Tianjun Zhang, Yu Zeng, Reetuparna Das, David Blaauw, and Satish Narayanasamy. 2018. GenAx: A genome sequencing accelerator. In *ISCA*.

[23] 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*.

[24] NCBI GenBank. 2022. U00096.3. https://www.ncbi.nlm.nih.gov/nuccore/U00096, visited 2025-01-08.

[25] Alice Maria Giani, Guido Roberto Gallo, Luca Gianfranceschi, and Giulio Formenti. 2020. Long walk to genomics: History and current approaches to genome sequencing and assembly. *Computational and Structural Biotechnology Journal* 18 (2020), 9–19.

[26] Sara Goodwin, John D McPherson, and W Richard McCombie. 2016. Coming of age: ten years of next-generation sequencing technologies. *Nature reviews genetics* 17, 6 (2016), 333–351.

[27] Osamu Gotoh. 1990. Optimal sequence alignment allowing for long gaps. *Bulletin of mathematical biology* 52, 3 (1990), 359–373.

[28] SAM/BAM Format Specification Working Group. 2021. *Sequence Alignment/Map Format Specification.* https://samtools.github.io/hts-specs/SAMv1.pdf Accessed: January 17, 2025.

[29] Licheng Guo, Jason Lau, Zhenyuan Ruan, Peng Wei, and Jason Cong. 2019. Hardware acceleration of long read pairwise overlapping in genome sequencing: A race between fpga and gpu. In *FCCM*.

[30] Brandon Harris, Arpith C Jacob, Joseph M Lancaster, Jeremy Buhler, and Roger D Chamberlain. 2007. A banded Smith-Waterman FPGA accelerator for Mercury BLASTP. In *FPL*.

[31] Ariane L Hofmann, Jonas Behr, Jochen Singer, Jack Kuipers, Christian Beisel, Peter Schraml, Holger Moch, and Niko Beerenwinkel. 2017. Detailed simulation of cancer exome sequencing data reveals differences and common limitations of variant callers. *BMC bioinformatics* 18 (2017), 1–15.

[32] Ernst Joachim Houtgast, Vlad-Mihai Sima, Koen Bertels, and Zaid Al-Ars. 2015. An FPGA-based systolic array to accelerate the BWA-MEM genomic mapping algorithm. In *SAMOS*.

[33] Ernst Joachim Houtgast, Vlad-Mihai Sima, Koen Bertels, and Zaid Al-Ars. 2018. Hardware acceleration of BWA-MEM genomic short read mapping for longer read lengths. *Computational biology and chemistry* 75 (2018), 54–64.

[34] Bhavna Hurgobin. 2016. Short read alignment using SOAP2. *Plant Bioinformatics: Methods and Protocols* (2016), 241–252.

[35] Youngmok Jung and Dongsu Han. [n. d.]. BWA-MEME: BWA-MEM emulated with a machine learning approach. 38, 9 ([n. d.]), 2404–2413. doi:10.1093/bioinformatics/btac137

[36] Saurabh Kalikar, Chirag Jain, Md Vasimuddin, and Sanchit Misra. 2022. Accelerating minimap2 for long-read sequencing applications on modern CPUs. *Nature Computational Science* 2, 2 (2022), 78–83.

[37] Changdae Kim, Kwangwon Koh, Taehoon Kim, Daegyu Han, and Jiwon Seo. 2022. BWA-MEM-SCALE: Accelerating genome sequence mapping on commodity servers. In *ICPP*.

[38] Ben Langmead and Steven L Salzberg. 2012. Fast gapped-read alignment with Bowtie 2. *Nature methods* 9, 4 (2012), 357–359.

[39] Heng Li. 2013. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. *arXiv preprint arXiv:1303.3997* (2013).

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

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

[42] Heng Li and Richard Durbin. 2009. Fast and accurate short read alignment with Burrows–Wheeler transform. *Bioinformatics* 25, 14 (2009), 1754–1760.

[43] Heng Li, Xiaowen Feng, and Chong Chu. 2020. The design and construction of reference pangenome graphs with minigraph. *Genome biology* 21 (2020), 1–19.

[44] Ruiqiang Li, Yingrui Li, Karsten Kristiansen, and Jun Wang. 2008. SOAP: short oligonucleotide alignment program. *Bioinformatics* 24, 5 (2008), 713–714.

[45] Wen-Wei Liao, Mobin Asri, Jana Ebler, Daniel Doerr, Marina Haukness, Glenn Hickey, Shuangjia Lu, Julian K Lucas, Jean Monlong, Haley J Abel, et al. 2023. A draft human pangenome reference. *Nature* 617, 7960 (2023), 312–324.

[46] Chi-Man Liu, Thomas Wong, Edward Wu, Ruibang Luo, Siu-Ming Yiu, Yingrui Li, Bingqiang Wang, Chang Yu, Xiaowen Chu, Kaiyong Zhao, et al. 2012. SOAP3: Ultra-fast GPU-based parallel alignment tool for short reads. *Bioinformatics* 28, 6 (2012), 878–879.

[47] Weiguo Liu, Bertil Schmidt, Gerrit Voss, Andre Schroder, and Wolfgang Muller-Wittig. 2006. Bio-sequence database scanning on a GPU. In *IPDPS*.

[48] Yang Liu, Wayne Huang, John Johnson, and Sheila Vaidya. 2006. GPU accelerated Smith-Waterman. In *ICCS*. Springer.

[49] Yongchao Liu and Bertil Schmidt. 2013. CUSHAW2-GPU: Empowering faster gapped short-read alignment using GPU computing. *IEEE Design & Test* 31, 1 (2013), 31–39.

[50] Yongchao Liu and Bertil Schmidt. 2015. GSWABE: Faster GPU-accelerated sequence alignment with optimal alignment retrieval for short DNA sequences. *Concurrency and Computation: Practice and Experience* 27, 4 (2015), 958–972.

[51] Yongchao Liu, Bertil Schmidt, and Douglas L Maskell. 2012. CUSHAW: A CUDA compatible short read aligner to large genomes based on the Burrows–Wheeler transform. *Bioinformatics* 28, 14 (2012), 1830–1837.

[52] Kisaru Liyanage, Hiruna Samarakoon, Sri Parameswaran, and Hasindu Gamaarachchi. 2023. Efficient end-to-end long-read sequence mapping using minimap2-fpga integrated with hardware accelerated chaining. *Scientific Reports* 13, 1 (2023), 20174.

[53] Ruibang Luo, Thomas Wong, Jianqiao Zhu, Chi-Man Liu, Xiaoqian Zhu, Edward Wu, Lap-Kei Lee, Haoxiang Lin, Wenjuan Zhu, David W Cheung, et al. 2013. SOAP3-dp: Fast, accurate and sensitive GPU-based short read aligner. *PloS one* 8, 5 (2013), e65632.

[54] Clemens Lutz, Sebastian Breß, Steffen Zeuch, Tilmann Rabl, and Volker Markl. 2022. Triton join: Efficiently scaling to a large join state on gpus with fast interconnects. In *SIGMOD*.

[55] Neil A Miller, Emily G Farrow, Margaret Gibson, Laurel K Willig, Greyson Twist, Byunggil Yoo, Tyler Marrs, Shane Corder, Lisa Krivohlavek, Adam Walter, et al. 2015. A 26-hour system of highly sensitive whole genome sequencing for emergency management of genetic diseases. *Genome medicine* 7 (2015), 1–16.

[56] André Müller, Bertil Schmidt, Richard Membarth, Roland Leißa, and Sebastian Hack. 2022. Anyseq/gpu: a novel approach for faster sequence alignment on gpus. In *Proceedings of the 36th ACM International Conference on Supercomputing*. 1–11.

[57] NCBI. 1982. GenBank. https://www.ncbi.nlm.nih.gov/genbank/, visited 2024-01-08.

[58] Saul B Needleman and Christian D Wunsch. 1970. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology* 48, 3 (1970), 443–453.

[59] NIST. [n. d.]. Genome in a Bottle. https://www.nist.gov/programs-projects/genome-bottle.

[60] NVIDIA. 2022. Boosting Dynamic Programming Performance Using NVIDIA Hopper GPU DPX Instructions. https://developer.nvidia.com/blog/boosting-dynamic-programming-performance-using-nvidia-hopper-gpu-dpx-instructions/, visited 2024-10-11.

[61] NVIDIA Corporation. 2024. NVIDIA Parabricks. https://www.nvidia.com/en-us/clara/parabricks/. Accessed: 2024-10-11.

[62] Seongyeon Park, Junguk Hong, Jaeyong Song, Hajin Kim, Youngsok Kim, and Jinho Lee. 2024. AGAThA: Fast and Efficient GPU Acceleration of Guided Sequence Alignment for Long Read Mapping. In *PPoPP*.

[63] Seongyeon Park, Hajin Kim, Tanveer Ahmad, Nauman Ahmed, Zaid Al-Ars, H Peter Hofstee, Youngsok Kim, and Jinho Lee. 2022. SALoBa: Maximizing Data Locality and Workload Balance for Fast Sequence Alignment on GPUs. In *IPDPS*.

[64] Johns Paul, Shengliang Lu, Bingsheng He, and Chiew Tong Lau. 2021. MG-Join: A scalable join for massively parallel multi-GPU architectures. In *SIGMOD*.

[65] Minh Pham, Yicheng Tu, and Xiaoyi Lv. 2023. Accelerating bwa-mem read mapping on gpus. In *ICS*.

[66] Mehdi Pirooznia, Melissa Kramer, Jennifer Parla, Fernando S Goes, James B Potash, W Richard McCombie, and Peter P Zandi. 2014. Validation and assessment of variant calling pipelines for next-generation sequencing. *Human genomics* 8 (2014), 1–10.

[67] MV Ramakrishna and Justin Zobel. 1997. Performance in practice of string hashing functions. In *Database Systems For Advanced Applications' 97*. World Scientific, 215–223.

[68] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *KDD*.

[69] NCBI RefSeq. 2019. GRCh38.p13. https://www.ncbi.nlm.nih.gov/data-hub/genome/GCF_000001405.39/, visited 2025-01-08.

[70] Eric Sven Ristad and Peter N Yianilos. 1998. Learning string-edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20, 5 (1998), 522–532.

[71] Michael Roberts, Wayne Hayes, Brian R Hunt, Stephen M Mount, and James A Yorke. 2004. Reducing storage requirements for biological sequence comparison. *Bioinformatics* 20, 18 (2004), 3363–3369.

[72] Torbjørn Rognes. 2011. Faster Smith-Waterman database searches with inter-sequence SIMD parallelisation. *BMC bioinformatics* 12 (2011), 1–11.

[73] Torbjørn Rognes and Erling Seeberg. 2000. Six-fold speed-up of Smith–Waterman sequence database searches using parallel processing on common microprocessors. *Bioinformatics* 16, 8 (2000), 699–706.

[74] Enzo Rucci, Carlos Garcia, Guillermo Botella, Armando De Giusti, Marcelo Naiouf, and Manuel Prieto-Matías. 2015. An energy-aware performance analysis of SWIMM: Smith–Waterman implementation on Intel's Multicore and Manycore architectures. *Concurrency and Computation: Practice and Experience* 27, 18 (2015), 5517–5537.

[75] Harisankar Sadasivan, Milos Maric, Eric Dawson, Vishanth Iyer, Johnny Israeli, and Satish Narayanasamy. 2023. Accelerating Minimap2 for accurate long read alignment on GPUs. *Journal of biotechnology and biomedicine* 6, 1 (2023), 13.

[76] Kristoffer Sahlin. 2022. Strobealign: flexible seed size enables ultra-fast and accurate read alignment. *Genome Biology* 23, 1 (2022), 260.

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

[78] Edans Flavius de O Sandes and Alba Cristina MA de Melo. 2011. Smith-Waterman alignment of huge sequences with GPU in linear space. In *IPDPS*.

[79] Edans Flavius O Sandes and Alba Cristina MA de Melo. 2010. CUDAlign: Using GPU to accelerate the comparison of megabase genomic sequences. In *PPoPP*.

[80] Bertil Schmidt, Felix Kallenborn, Alejandro Chacon, and Christian Hundt. 2024. CUDASW++ 4.0: ultra-fast GPU-based Smith–Waterman protein sequence database search. *BMC bioinformatics* 25, 1 (2024), 342.

[81] Markus Schütz, Bernhard Kerbl, and Michael Wimmer. 2022. Software rasterization of 2 billion points in real time. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 5, 3 (2022), 1–17.

[82] Ole Seehausen, Roger K Butlin, Irene Keller, Catherine E Wagner, Janette W Boughman, Paul A Hohenlohe, Catherine L Peichel, Glenn-Peter Saetre, Claudia Bank, Åke Brännström, et al. 2014. Genomics and the origin of species. *Nature Reviews Genetics* 15, 3 (2014), 176–192.

[83] Jay Shendure, Shankar Balasubramanian, George M Church, Walter Gilbert, Jane Rogers, Jeffery A Schloss, and Robert H Waterston. 2017. DNA sequencing at 40: past, present and future. *Nature* 550, 7676 (2017), 345–353.

[84] Carola Simon and Rolf Daniel. 2011. Metagenomic analyses: past and future trends. *Applied and environmental microbiology* 77, 4 (2011), 1153–1161.

[85] Temple F Smith, Michael S Waterman, et al. 1981. Identification of common molecular subsequences. *Journal of molecular biology* 147, 1 (1981), 195–197.

[86] Jang-il Sohn and Jin-Wu Nam. 2018. The present and future of de novo whole-genome assembly. *Briefings in bioinformatics* 19, 1 (2018), 23–40.

[87] 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*.

[88] Gincy Paily Thottathil, Kandakumar Jayasekaran, and Ahmad Sofiman Othman. 2016. Sequencing crop genomes: a gateway to improve tropical agriculture. *Tropical life sciences research* 27, 1 (2016), 93.

[89] 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 *HPCA*.

[90] Geraldine A Van der Auwera, Mauricio O Carneiro, Christopher Hartl, Ryan Poplin, Guillermo Del Angel, Ami Levy-Moonshine, Tadeusz Jordan, Khalid Shakir, David Roazen, Joel Thibault, et al. 2013. From FastQ data to high-confidence variant calls: the genome analysis toolkit best practices pipeline. *Current protocols in bioinformatics* 43, 1 (2013), 11–10.

[91] Md Vasimuddin, Sanchit Misra, Heng Li, and Srinivas Aluru. 2019. Efficient architecture-aware acceleration of BWA-MEM for multicore systems. In *IPDPS*.

[92] Lipeng Wang, Yuandong Chan, Xiaohui Duan, Haidong Lan, Xiangxu Meng, and Weiguo Liu. 2014. Xsw: Accelerating biological database search on xeon phi. In *2014 IEEE International Parallel & Distributed Processing Symposium Workshops*. IEEE, 950–957.

[93] Lifeng Yan, Zekun Yin, Jinjin Li, Yang Yang, Tong Zhang, Fangjin Zhu, Xiaohui Duan, Bertil Schmidt, and Weiguo Liu. 2024. RabbitSAlign: Accelerating Short-Read Alignment for CPU-GPU Heterogeneous Platforms. In *International Symposium on Bioinformatics Research and Applications*. Springer, 83–94.

[94] Zhen Yao, Frank M You, Amidou N'Diaye, Ron E Knox, Curt McCartney, Colin W Hiebert, Curtis Pozniak, and Wayne Xu. 2020. Evaluation of variant calling tools for large plant genome re-sequencing. *BMC bioinformatics* 21 (2020), 1–16.

[95] Alberto Zeni, Giulia Guidi, Marquita Ellis, Nan Ding, Marco D Santambrogio, Steven Hofmeyr, Aydın Buluç, Leonid Oliker, and Katherine Yelick. 2020. LOGAN: High-performance GPU-based x-drop long-read alignment. In *IPDPS*. New Orleans, LA, USA.

[96] Haitang Zhang, Junchao Ma, Zixia Qiu, Junmei Yao, Mustafa A Al Sibahee, Zaid Ameen Abduljabbar, and Vincent Omollo Nyangaresi. 2023. Multi-GPU Parallel Pipeline Rendering with Splitting Frame. In *CGI*.

[97] Peiheng Zhang, Guangming Tan, and Guang R Gao. 2007. Implementation of the Smith-Waterman algorithm on a reconfigurable supercomputing platform. In *International Workshop on High-performance Reconfigurable Computing Technology and Applications*.

[98] Zhonghai Zhang, Yewen Li, Ke Meng, Chunming Zhang, and Guangming Tan. 2025. FastBWA: Practical and Cost-Efficient Genome Sequence Alignment Pipeline. In *Proceedings of the 30th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*. 554–556.

[99] Anbo Zhou, Timothy Lin, and Jinchuan Xing. 2019. Evaluating nanopore sequencing data processing pipelines for structural variation identification. *Genome biology* 20 (2019), 1–13.