SeedEx: A Genome Sequencing Accelerator for Optimal Alignments in Subminimal Space

Daichi Fujiki Shunhao Wu Nathan Ozog Kush Goliya
David Blaauw Satish Narayanasamy Reetuparna Das
University of Michigan
{dfujiki, shunhao, ozog, kgoliya, blaauw, nsatish, reetudas}@umich.edu

***Abstract*—Innovations in genome sequencing techniques are enabling remarkably fast and low cost production of raw genome data. As Moore's law tapers off, bottlenecks in genome sequencing are shifting to computational resources for mapping reads to reference DNA. This paper presents SeedEx, a read-alignment accelerator focused on the seed-extension step. SeedEx is based on the observation that only a small fraction of reads require large edit distance for alignment, hence an area efficient narrow-band seed-extension accelerator can suffice in practice. However, due to the highly error-sensitive nature of genomic workloads, guaranteeing optimality of alignment result is of cardinal importance. Towards this end, we propose a speculation-and-test based framework by using strict but powerful optimality checking mechanisms. We demonstrate SeedEx by an implementation on a cloud FPGA. SeedEx achieves 6.0× iso-area throughput speedup when compared to a banded Smith-Waterman baseline, and achieving 43.9 M seed extentions/s on AWS `f1.2xlarge` instance. Integration with BWA-MEM2 improves the execution time by 2.3×.**

***Index Terms*—genome sequencing, accelerator, FPGA**

I. Introduction

Next generation health care is enabled by genomics and precision medicine. The recent remarkable growth in computational horsepower and sequencing equipment unlocks the potential to transform personalized medicine. For example, understanding mutations in the cancer cell of a particular patient makes it possible to devise individualized treatment plans [1], [2]. Also, large-scale genome analyses of diverse populations have furnished clues to understanding the causes of various diseases ranging from cancer [3], Alzheimer's [4], to rare genetic disorders [5]. It also facilitates assessing risk factors and developing better cures.

The human genome consists of a long sequence of DNA base pairs (bps) and has 3.08 Giga bps of nucleotides (A, G, T, and C). DNA specimen is first fragmented into billions of short DNA sequences (called *reads*) by a sequencing instrument. After this primary analysis procedure, secondary analysis analyzes the reads to align them to the reference genome. This task is complicated as an individual's genome may not exactly match the reference genome, and the end goal is to determine the variants in the new genome. Furthermore, the sequencing machine can introduce errors into the reads as well. To address this problem, the sequencing machine produces several reads (30× - 50×) to cover every position in the genome, requiring more computation.

Currently, whole genome secondary analysis takes tens to hundreds of CPU hours [6]. While there are several computational steps in sequencing raw genome data, we focus on accelerating *read alignment*, a dominant step in the secondary analysis. Read alignment determines the position of a read in the genome. Due to variants and sequencing errors, a read (referred to as query $\mathtt{Q}$) may not perfectly match a substring in the reference genome $\mathtt{R}$. Sequence aligners solve this problem in two steps: *seeding* and *seed-extension*. Seeding finds perfect matches in the reference genome for small substrings (*seeds*) in a read. The seed positions are then *extended* to determine the best position via an approximate string matching algorithm.

While solutions to approximate string matching for seed extension have been widely studied, the most commonly used algorithm, particularly in genomics, is a dynamic programming algorithm called Smith-Waterman [7]. It computes the edit distance between two strings by filling a grid of size N^2, where N is the string length. A hardware rendition of Smith-Waterman can use a banded implementation, where only cells within a band of size w around the principal diagonal of the Smith-Waterman matrix are computed with w Processing Elements (PE's) achieving best throughput. A large w is required to capture maximum possible edits between the query and reference [8] and guarantee that the optimal alignment is found [6].

The area consumption of an accelerator with w PEs is high for a w which achieves optimal alignment. Area efficiency is especially critical for FPGA-based acceleration, where market volume does not justify ASIC development cost [9]. While there have been several optimizations [10], [11], including hardware accelerators [12]–[16], they either require PEs as large as the string length for $\mathcal{O}(N)$ latency, or give up optimality guarantee. The optimality guarantee is important for two reasons. *First*, accurately capturing variants between genomes is the ultimate goal of genome sequencing. The differences in human genomes are minuscule, about 0.1%. Thus even small errors in alignment can lead to expensive clinical mistakes in critical disease diagnosis. *Second*, ensuring bit equivalence to standard aligners. This is important because several research and clinical applications rely on well-known, clinically validated, behavior of alignment tools on a sufficiently large number of biological samples. Even small, seemingly innocuous differences in output could require costly

978-1-7281-7383-2/20/$31.00 ©2020 IEEE
DOI 10.1109/MICRO50266.2020.00080

re-validation of pipelines (e.g., FDA approvals) and becomes a barrier to practical usability. Recently introduced BWA-MEM2 [17] adheres to this policy and guarantees to produce a bit-equivalent result as BWA-MEM.

In this paper, we propose SeedEx, a hardware architecture which embodies a **speculation-and-test** based approach for seed extension. Our detailed workload analysis using real human genomes indicates that the majority of reads require a narrow band. Thus a small number of PEs can trace the optimal alignment for the majority of the inputs. However, since it is impossible to know the minimum required PEs prior to execution, accelerators need to implement a large set of PEs. This causes low utilization, leading to significantly lower throughput per unit area. SeedEx addresses this problem by allowing reads to be speculatively executed on a hardware with a narrow band accelerator. At the same time, SeedEx is able to grant optimality by introducing strict, but powerful, optimality checking mechanisms.

The proposed accelerator architecture consists of several SeedEx accelerator tiles. Each SeedEx accelerator tile consists of a hierarchy of banded Smith-Waterman systolic array machines. The first level of the hierarchy consists of narrow-band affine gap scoring machines, which require complex calculations to support the affine gap. The second level consists of light-weight edit machines, which do not have weighted penalty and can be implemented with low-complexity delta-encoding. The edit machine helps to improve optimality check for input with asymmetric string lengths of query and reference strings. SeedEx tests the optimality by first comparing the score from a narrow-band affine-gap machine to upper bound scores. If these checks fail, the query and reference are sent to an edit machine, followed by a comparison of narrow-band scores and optimistic edit scores. The small number of reads which fail all SeedEx tests are sent back from the accelerator to host CPU for rerun.

We implement SeedEx architecture on a cloud-based FPGA and provide complete integration with the host CPU. We verify SeedEx with FPGA and ensure bit equivalence of output with Broad Institute's BWA-MEM software [6] for read alignment. Note that SeedEx offers a modular design and generic interface which can be integrated into different aligners other than BWA-MEM.

In summary, this paper makes the following contributions:

- We conduct a detailed analysis of required band size for the Smith-Waterman algorithm with human genomic data and observe that only a small fraction of inputs require a large band. We leverage this to improve area efficiency and design a narrow-band accelerator requiring fewer PEs.
- We present SeedEx architecture, a hierarchical architecture consisting of narrow-band affine-gap machines and optimistic edit machines.
- SeedEx incorporates a **speculation-and-test** based mechanism that guarantees optimality[1]. The testing mechanisms incur 5.53% area overhead over a narrow band machine while allowing 98% of the inputs to be processed on the accelerator. The remaining 2% are rerun on the host.
- We implement SeedEx architecture for a cloud FPGA. SeedEx achieves 6.0× iso-area throughput speedup compared to banded Smith-Waterman baseline. SeedEx achieves 43.9 M seed extensions/s on AWS `f1.2xlarge` instance.
- SeedEx integrated with a seeding accelerator achieves speedups of 3.75× over BWA-MEM and 2.28× over BWA-MEM 2 on an AWS FPGA instance.

II. Preliminary Analysis

A. Background

Read alignment is one of the time-consuming steps in genome sequencing. This process is responsible for determining the best candidate position of a read in the reference genome. Read aligners solve this problem in two steps: seeding and seed-extension. Seeding finds perfect matches in the reference genome for small sub-strings (seeds) in a read. The seed-extension step then extends these seed positions to determine the best position by using an approximate string matching algorithm based on a dynamic programming (DP) algorithm [18], [19].

The most widely used DP algorithms particularly designed for genomic seed extension are Smith-Waterman [7] and Needleman-Wunsch [20]. These algorithms compute a similarity score between two strings by filling a matrix of size N^2, where N is the string length. The similarity score is generated using an affine gap function [21], [22], which is based on edit distance but weighs different edit types separately. We define an affine gap scoring $\mathsf{s}_{af} = \{m, x, g_o, g_e\}$, where m, x, g_o, and g_e denote the match reward, the mismatch penalty, the gap opening penalty, and the gap extension penalty, respectively. Based on s_{af}, the affine gap function calculates a score $H_{i,j}$ for a DP cell (i, j) as

$$H_{i,j} = \max\{H_{i-1,j-1} + S_{i,j}, E_{i,j}, F_{i,j}\}, \quad (1)$$

$$E_{i+1,j} = \max\{H_{i,j} - g_o, E_{i,j}\} - g_e, \quad (2)$$

$$F_{i,j+1} = \max\{H_{i,j} - g_o, F_{i,j}\} - g_e. \quad (3)$$

Here $S_{i,j} = m$ if reference at position i matches query at j, else x. The reference position for a read that yields the highest score is chosen as that read's mapping position.

The final output also contains a trace of edits needed to align the read to the reference string at the chosen reference position. This final step is referred to as traceback, which constructs the trace with the optimal alignment by tracing back pointers starting from the highest-scoring cell. Supporting traceback in an accelerator can be expensive and complex [8], [23]. We observe that a trace is reported only for the highest-scoring

[1]Optimality is guaranteed with respect to reference string input to Smith-Waterman matrix, targeting global and semi-global alignments.

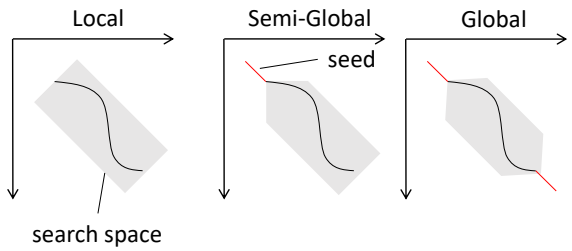

Fig. 1. Local, semi-global, and global alignment. x- and y-axis correspond to query and reference string, and the solid line shows an alignment result (trace).

position [6], thus it can be done once per read on the host rather than for every seed-extension in a read on the accelerator (≈10 extensions per read).

Smith-Waterman is used for local alignment as in Figure 1 (left). In local alignment, gaps at both of the ends of the read are not penalized, and the result trace is given based on the maximum score obtained by aligning a substring of the read. On the other hand, Needleman-Wunsch penalizes gaps at the ends and produces an end-to-end alignment, as shown in Figure 1 (right). This is called global alignment. Furthermore, there exists an approach called semi-global alignment, where the algorithm penalizes only gaps at one end and does not penalize the other end, as shown in Figure 1 (middle). This is the key seed extension kernel in the mainstream aligners, including BWA-MEM and BWA-MEM2. SeedEx targets the semi-global and global alignment.

Several approaches have been explored to limit the cells in the N^2 DP table filled by the alignment algorithms, including fixed/adaptive banding [23]–[27] and search based pruning [28]–[31]. For example, prior works use classic best-first search algorithms, such as the A* search algorithm [30], [31]. With an admissible heuristic that estimates the cost of the optimal path to the end, A* is guaranteed to return the optimal path with minimal exploration space where the optimality can be inferred by the heuristic. However, the cost to implement A* search is prohibitively high in both hardware and software due to the complexity of priority queues and the difficulty in parallelization.

Banded algorithms calculate scores in cells on w band around the main diagonal running from upper-left to lower-right. These banding based approaches offer efficiency because the computational complexity is reduced from $\mathcal{O}(N^2)$ to $\mathcal{O}(Nw)$, however, they have difficulty in guaranteeing optimality of the generated alignment for a small w.

B. Analysis of Banded Algorithms

Banded algorithms enhance the efficiency of both software and hardware due to reduced computation complexity. State-of-the-art alignment tools such as BWA-MEM [6], [17] adjust the band dynamically in software for each seed extension. In addition, the maximum band size is estimated before running an extension. It is calculated by taking the largest of the maximum possible insertions and deletions of a given query. However, the estimated band is proportional to the query

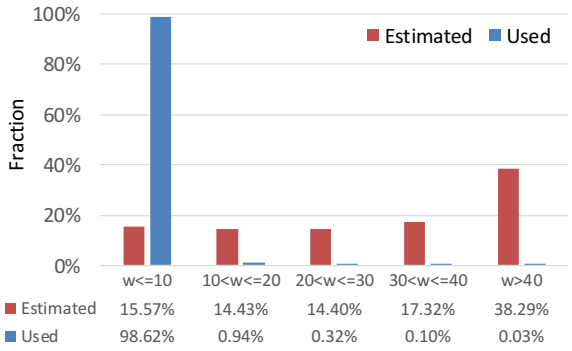

Fig. 2. Band distribution of BWA-MEM [6], [17].

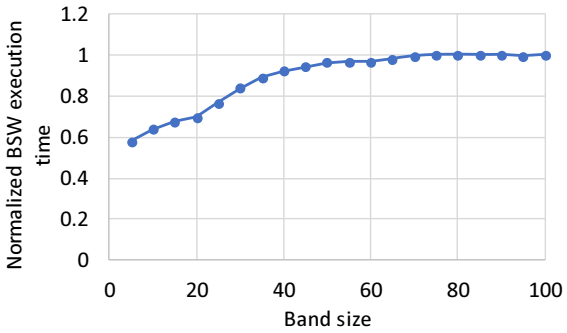

Fig. 3. Band vs. seed extension execution time.

length and is very conservative, not to miss the optimal scores. We define this maximum band as *full-band*.

Figure 2 shows the distribution of band both *estimated* and *used* by BWA-MEM. The band size is represented as w. We observe that a band size of $w > 40$ is estimated for more than 38% of the extensions. However, more than 98% of the extensions need $w \leq 10$, and a large band is required for just remaining 2% of the extensions. This motivates the use of a narrow-band accelerator, which can safely execute seed-extensions only if there is a mechanism to check for optimality.

In Figure 3, we plot band size vs. execution time for the banded Smith-Waterman (BSW) kernel in an unmodified version of BWA-MEM2. A smaller band reduces inner-loop iterations of the software, and its effect can be observed in the execution time. Due to the kernel's early termination optimizations, execution time saturates as the band grows. As a result, the conservative band does not degrade performance significantly for software.

On the other hand, hardware accelerators can enjoy direct benefits from a narrow band, which can reduce the required number of PEs. We design a systolic array based BSW accelerator and plot its area tradeoff with the band in Figure 4. While a narrow-band accelerator is promising, we cannot reap considerable benefits because of the discrepancy between (a) apriori conservative estimations of band size (Figure 2 - *Estimated*), and (b) the posterior reality band size (Figure 2 - *Used*). Our work proposes a way out of this dilemma, by granting narrow-band processing units (PUs) the ability

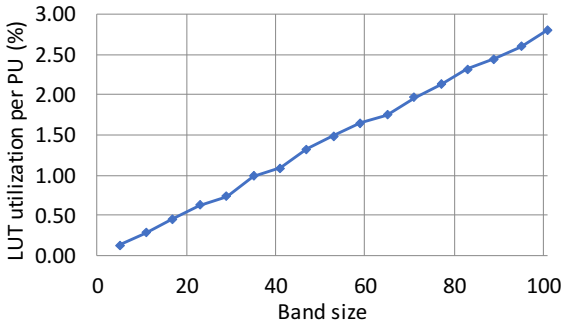

Fig. 4. Band vs. hardware resources of accelerator.

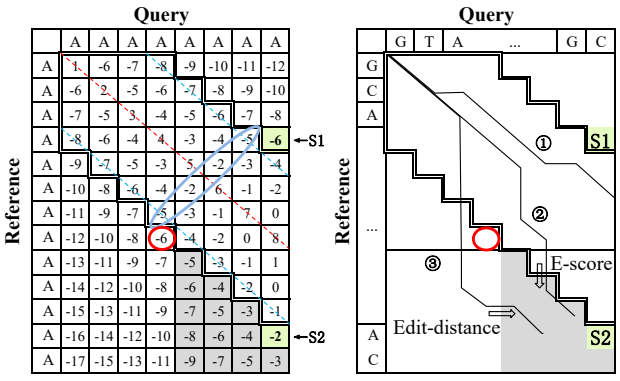

Fig. 5. Optimality check matrix and thresholds.

to guarantee optimality using a speculation-and-test based approach.

III. SeedEx Optimality Checks

SeedEx follows a speculation-and-test based approach to guarantee the optimality of a narrow band. The proposed method manifests the *necessary conditions* for the score generated within the narrow band to be optimal. In other words, the method confirms the non-existence of the optimal score outside the region in the band. It is important to note that the proposed method does not alter the dynamic programming algorithm or its hardware/software implementation. Rather, logic is added to test whether the score generated is optimal or not. We introduce three checks, a thresholding mechanism, an E-score check, and an edit-distance check.

A. Thresholding

The thresholding mechanism is used to calculate the theoretical highest score (upper-bound score) outside the band and compare it against the score obtained within the band. If the score obtained within the band is strictly higher than the theoretical highest score outside, the optimality of the narrow band score is guaranteed. Otherwise, there could be a path outside the band leading to a better score, and additional checks are required. We will prove the following theorem by providing a way to calculate the threshold scores. The theorem is true for both global and semi-global alignments.

Theorem 1. *There exist theoretical upper-bound scores (threshold scores) such that any alignment with a score greater than upper-bound scores is guaranteed to be optimal even if generated by a narrow-band algorithm.*

To calculate the theoretical upper-bound scores, we build a score matrix assuming best alignment (i.e., all matches) and determine the maximum possible score for each cell. As shown in Figure 5, we use two thresholds, due to the asymmetry between query length and target length. The first threshold, $S1$, is used to ensure no better score can be obtained above the narrow band, while the second threshold, $S2$, is for below the narrow band. Note that the theoretical highest score at the red circle is the same as $S1$. Generally, $S2$ is a stricter threshold than $S1$, because there are more matches when calculating $S2$, leading to a higher score. In the example of Figure 5, the narrow band is shown with the dashed lines.

The score thresholds can be derived as a function of a scoring method used by the sequence alignment algorithm, query length N, and a band parameter w. For semi-global alignment, the scoring thresholds are calculated as follows:

$$S1 = h_0 - [g_0 + wg_e] + [N - w]m, \tag{4}$$

$$S2 = h_0 - [g_0 + wg_e] + Nm, \tag{5}$$

where m is match reward, g_o is gap opening penalty, g_e is gap extension penalty, and h_0 is initial score from a seed (optional). Essentially, $S1$ is calculated by adding the seed score h_0, subtracting the penalty for a gap in the Query $g_0 + wg_e$, and adding the score assuming the rest of the characters match $[N-w]m$. It is similar for calculating $S2$; the only difference is the number of characters assumed matching is N instead of $N - w$. The formulation above can be easily extended for global alignment by replacing g_o with $2g_o$ and g_e with $2g_e$.

There are three cases regarding the relation between the best score within the band, $score_{nb}$, and the thresholds. (a) If $score_{nb}$ fails $S1$ ($score_{nb} \leq S1$), it is extremely small. We simply rerun the extension on software with a full-band. (b) If $score_{nb}$ passes $S2$ ($score_{nb} > S2$), the optimality is guaranteed. (c) If $score_{nb}$ is in between $S1$ and $S2$ ($S1 < score_{nb} \leq S2$), the optimality is not guaranteed, but additional checks can be applied to ensure optimality and avoid a rerun.

B. $Score_{outside}$

Before explaining the additional optimality checks, it is essential to establish where the optimal score might be obtained. When $score_{nb}$ is in between $S1$ and $S2$, the best score outside the narrow band, $score_{outside}$, could be better than $score_{nb}$, and the highest between the two is the true optimal score. Assume $score_{outside}$ is the higher one, then it must be obtained in the shaded region in the DP matrix in Figure 5.

Lemma 2. *If $S1 < score_{nb} \leq S2$, and if $score_{outside} > score_{nb}$, then $score_{outside}$ must be obtained in the shaded region in Figure 5.*

Proof. It can be proven with a contradiction approach. Assume $score_{outside}$ is obtained in the non-shaded region outside the

band, then $score_{outside} \leq S1$, since $S1$ is the theoretical highest score in the non-shaded region outside the band. It is known that $S1 < score_{nb} \leq S2$, so $score_{outside} < score_{nb}$, which contradicts the assumption that $score_{outside}$ is better than $score_{nb}$. Therefore, if $score_{outside}$ is higher than $score_{nb}$, it must be obtained in the shaded region. □

The goal of the additional checks is to examine whether the best score outside the narrow band, $score_{outside}$ is higher than $score_{nb}$. Given $score_{outside}$ can only be obtained in the shaded region, it is obvious that the path leading to the score can either come from the top of the shaded region (path ② in Figure 5), or from the left (path ③ in Figure 5).

C. E-score Check

The E-score check is designed to eliminate the possibility that there is a path entering the shaded region in Figure 5 from the top, resulting in a higher score outside the band (path ②). E-score is referring to the E channel in the Dynamic Programming (DP) algorithm (Equation 2). For a given cell in the DP matrix, the E-score represents the best score coming from the cell above.

The steps for the E-score check are as following: for each cell on the top boundary of the shaded region, obtain the E-score from the narrow band, and assume all characters match for the rest of the query and add the matching score. Then, find the max among all the cells' score on the top boundary; this score, $score_{Max\_E}$, would be an optimistic estimate of the highest score obtained through a path coming from the top boundary into the shaded region. The score can be formulated as:

$$score_{Max\_E} = \max_{i \in [1,n]} \{E\text{-}score_i + (n - i + 1)m\}, \quad (6)$$

where n is the number of cells on the top boundary of the shaded region, $E\text{-}score_i$ is the E-score of the i^{th} cell on the top boundary of the shaded region starting from the top left, m is the match reward, and $n - i + 1$ represents the number of assumed matches for the rest of the query.

Finally, compare the optimistic $score_{Max\_E}$ against the best score within the band, $score_{nb}$. If $score_{Max\_E} < score_{nb}$, no path coming from the top boundary of the shaded region will result in a better score than $score_{nb}$, and the E-score check passes. Otherwise, the E-score check fails, and a rerun is needed.

D. Edit-Distance Check

After E-score check passes, there is still possibility that a path coming from the left boundary of the shaded region can result in a better score than $score_{nb}$ (path ③ in Figure 5), which is where the Edit-distance check comes into play. It is to perform an extra seed extension on the shaded region using $S1$, which is the same as the theoretical highest score at the circle, as the initial score. *While the same affine gap scoring scheme can be used for calculation, we use an optimistic scoring scheme such as edit-distance because it significantly lowers hardware complexity (Section IV-B).*

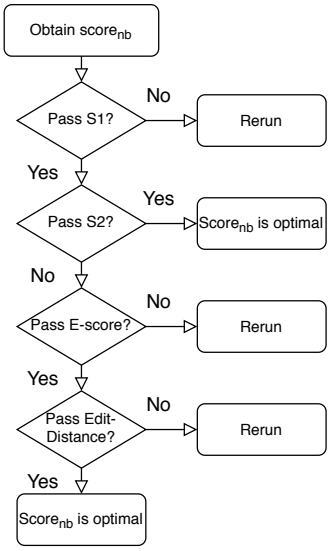

Fig. 6. SeedEx workflow.

The extra extension is essential to estimate the best score in the shaded region if we were to run the initial seed extension with a full-band. By using the theoretical upper bound $S1$ as the initial score, and using a lower gap penalty of edit-distance scoring, the extension score, $score_{ed}$, would be an optimistic estimate of the highest score obtained through a path coming from the left boundary into the shaded region. If $score_{nb} > score_{ed}$, meaning the narrow band score is better than the optimistic edit-distance score, then no path coming from the left boundary of the shaded region can result in a higher score, and the Edit-distance check passes. Otherwise, the check fails, and a rerun is needed.

E. Checks Workflow

If the best narrow-band score passes the stricter threshold $S2$, or if it is between $S1$ and $S2$ and passes both the E-score check and the Edit-distance check, its optimality is guaranteed. There cannot be a higher score obtained with a larger band. If the checks fail, the seed extension is rerun on the host CPU with the full-band estimated by BWA-MEM. Figure 6 is a summary of the workflow of the checks.

IV. SeedEx Architecture

In this section, we illustrate our architecture and hardware-specific optimizations of SeedEx. Figure 7 shows the top-level architecture of the SeedEx accelerator, which consists of an input query fetching interface and several SeedEx Cores. The input queries from memory interface are buffered and parsed, then loaded to a SeedEx core. Each SeedEx core consists of input parsing control logic, a few Banded Smith-Waterman (BSW) cores, and an Edit Machine core. Input sequences are chunked and sequentially fed from Input RAM to BSW Core by Arbiter and State Manager, which are capable of bookkeeping multiple input streams. BSW core performs banded Smith-Waterman and generates narrow-band score

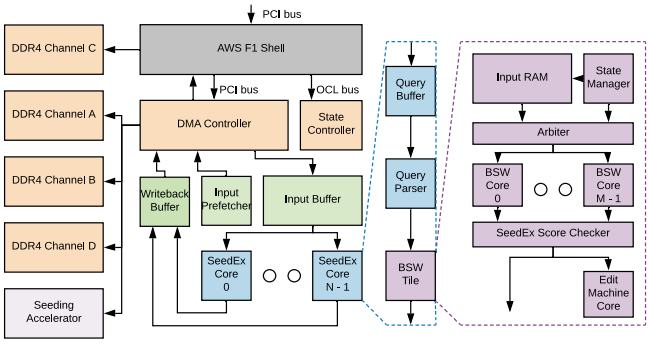

Fig. 7. SeedEx architecture.

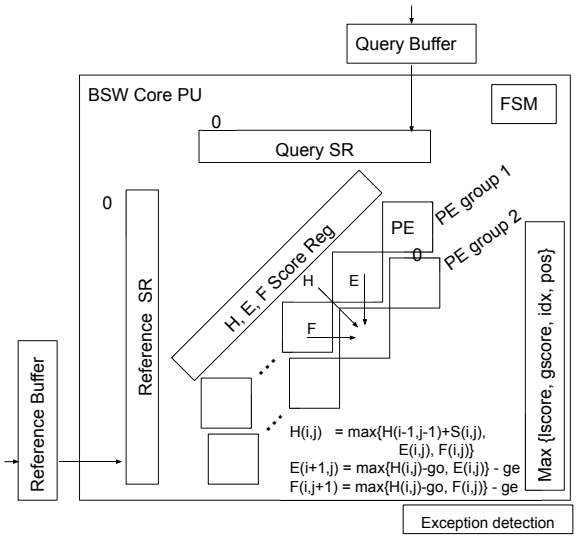

Fig. 8. BSW Core architecture.

$score_{nb}$ and E-check score $score_{Max\_E}$. If the narrow-band score passes the first threshold check but fails the second one (i.e. in between $S1$ and $S2$), and passes the E-score check, the query is sent to the edit machine core. The edit machine core generates the optimistic edit score $score_{ed}$. Recall that the SeedEx algorithm is a speculation based algorithm; it assumes a small band is sufficient for input problems and validates the outcomes by performing score validation checks, as shown in Figure 6. The remainder of this section will cover the main components of SeedEx Core, i.e., BSW core and Edit Machine core.

A. BSW Core

The BSW Core Engine is a systolic array implementation of the BSW DP algorithm. Figure 8 shows the block diagram of a BSW Core. An input genome string pair in a 3-bit format is loaded to the buffer and sequentially fed into a Query/Reference shift register (SR) one character pair at a time. When a PE observes valid inputs at its corresponding SR entries, it calculates cell score H, next vertical score input E, next horizontal score input F, as shown in Figure 8. While data in the SRs is only moving, one can virtually relate this to the PE groups marching along the main diagonal of the matrix every cycle. The scores for the next iteration are stored in the score registers. The initial scores are propagated to the score registers through PE's score E channel and score F channel using a special input symbol. This progressive initialization avoids long wires that are otherwise needed to provide connectivity across the large PE vectors.

The updated score is reported to score accumulators. Local score (lscore) accumulator records PE-row-wise maximum scores each cycle with its index in the matrix, and uses a shift register to reduce it to a single local score in parallel with the accumulation so as to minimize wiring. Global score (gscore) accumulator records scores when a PE crosses the right edge of the matrix.

While BWA-MEM fills the DP matrix row-by-row, it can stop score calculation for a row before reaching the band boundary. This early termination optimization is triggered when more than two consecutive cells in the previous row have zero H and E scores and the row-end is reached without observing a positive score afterward. Implementing this optimization is not straightforward in the systolic array. Because it processes multiple rows at a time, observing 2 zero scored cells within a row does not guarantee no positive score will show up from the rows above, of which computation is in progress. To generate bit-equivalent output with regard to BWA-MEM, we speculatively terminate any rows with more than two consecutive zero scores, and raise an exception when a positive score flows in from the cells above. Since such cases are extremely rare, extensions with this exception flag set are rerun on the CPU.

B. Edit Machine

Edit machine is a key source of efficiency which provides a super light-weight additional score check functionality to boost the passing rate of SeedEx check from 72% to 98%. As we discussed in Section III-D, the edit machine score check is done to guarantee the non-existence of a better score in the lower trapezoid region (Figure 5) not computed by the narrow band. While the original affine gap scoring scheme can be used for the lower trapezoid region, we choose to implement edit machine because of lower hardware complexity. Edit distance ($s_{ed} = \{$`m:1, x:-1, go:0, ge:-1`$\}$) (a.k.a Levenshtein distance) will always be greater than BWA-MEM's affine gap score $s_{af} = \{$`m:1, x:-4, go:-6, ge:-1`$\}$.

However, the potential benefit of the use of a naive edit distance machine is merely the reduction of register file area for E and F scores needed for affine gap calculation. Now, we introduce several optimizations to drastically reduce the area for edit distance calculation.

Delta Encoding: Datapath width is a principal determinant of the area of the edit machine. The naive implementation of an edit core needs to use an 8-bit data bus for calculating and storing data, as with the affine gap PU. This is ruled by the data size of the initial score and the dynamic range of score calculation. We adopt a coding technique in Lipon

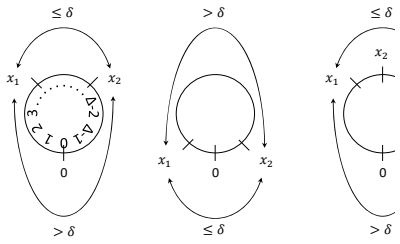

Fig. 9. Modulo circles [32] for delta max calculatio

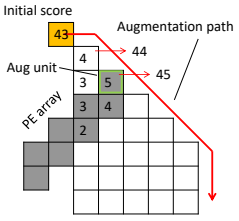

Fig. 10. Score decoding along augmentation path.

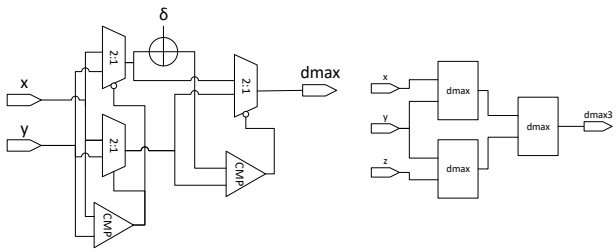

Fig. 11. 3 input delta max unit.

et al. [32], [33], and design 3-bit delta encoding PEs. While Lipton's scheme only supports global edit score, we revise it to support local score and scoring schemes for a 3-input Smith-Waterman PE.

Delta encoding is based on an insight that the scores of a cell in a DP table is typically bounded by a fixed dynamic range defined by the DP algorithm, which enables us to use residue arithmetic to calculate min/max. For example, assuming $h_{(i,j)}$ is the score for cell (i, j) in edit distance calculation, $h_{(i+1,j+i)}$ will be at least $h_{(i,j)} - 2$ (insertion + deletion) and at most $h_{(i,j)}+1$ (match). Now, we think about two candidates, X_1 and X_2, where larger of them is assigned to $h_{(i+1,j+i)}$. Based on the maximum difference $\delta = \max|X_1 - X_2| = 3$, we define a modulo circle [32] of which circumference is $\Delta \geq 2\delta+1$. Figure 9 (left, middle) shows two possible positional relationships of two residue values $(x_1, x_2) = (X_1, X_2) \mod \Delta$ (*wlog*, we assume $x_1 \leq x_2$) in the modulo circle. Because of the bounding constraints of X ruled by δ, whichever x precedes on the *small* arc of the Δ modulo circle in the clockwise rotation indicates the larger in the magnitude relationship of X_1 and X_2. Thus, assuming $x_1 \leq x_2$, if the small arc does not cross 0, X_2 is larger, otherwise X_1 is larger.

We design a 3-input delta max (dmax) unit using a 2-input dmax unit of $\Delta = 8$ as a building block (Figure 11). 3-input delta encoding redefines $\delta = \max|X_1-X_2, X_2-X_3, X_3-X_1|$ (Figure 9(right)).

The delta encoded scores are decoded by an augmentation unit, which reads scores along the augmentation path (Figure 10) starting from the initial score. It takes the residue value of full data-width input, compares it against the encoded value, and decodes it based on the positions in the modulo circle. An augmentation unit is connected to a PE, and the trajectory of PE with the augmentation unit in the matrix defines the augmentation path. Importantly, the augmentation path imposes a limitation of delta encoding. That is, any PE without an augmentation unit cannot translate a delta score into the actual score. Since we need to guarantee any scores in the trapezoid region are smaller than the narrow band score, reading out the edge scores on the path as in Figure 10 is not sufficient. However, combining every PE with an augmentation unit is expensive and spoils all the benefits of the reduced datapath. To combat this problem, we use a *relaxed* edit distance scoring $\mathsf{s}_{r\_ed} = \{\texttt{m:1, x:-1, go:0, ge(ins):0, ge(del):-1}\}$. By allowing zero penalty insertion, local scores are propagated in the horizontal direction and eventually read out by the single augmentation unit on the augmentation path. s_{r_ed} is still admissible and does not interfere with the delta encoding constraints.

Half-Width PE Array: In contrast to the BSW Core PU, Edit PU is not required to sweep a matrix of rectangular shape; rather, it sweeps half of it as shown in Figure 10. This allows us to design a half-width PE array, effectively reducing the PE area to 1/2.

V. System Design

In this section, we present the details of integration with a cloud-based FPGA. Subsequently, we illustrate the software architecture and the execution flow.

A. SeedEx Cloud FPGA Architecture

For the deployment of SeedEx infrastructure, we use the AWS EC2 F1 instance [34]. AWS enables the loading of custom logic, onto remote physical hardware, via an FPGA image. Figure 7 illustrates our top-level architecture integrated with AWS shell.

The custom logic (CL) architecture is instantiated in conjunction with AWS's shell interface. This interface consists of four AXI4 DDR4 memory channels, each with 16GiB. A PCI-e x16 link enables CL to Host communication. The SeedEx architecture was designed such that each memory channel can work independently. Host communication goes to a master state controller, which manages each memory channel's SeedEx cores. This state controller determines which regions of memory have been assigned for SeedEx input and output data, along with monitoring batch status.

Each SeedEx cluster on a single memory channel is comprised of four SeedEx clients. The clients-per-cluster ratio is chosen to strike a balance between memory bandwidth and area utilization. We employ prefetching of input queries to overlap input access latency (40 cycles on AWS AXI 4) with compute latency (≈100 cycles). Prefetched elements for an entire memory channel are stored in a block ram (BRAM) input buffer at the memory line granularity of 512-bits. After computation, results are coalesced into an output memory line at a five to one ratio before being written back to memory in a bandwidth efficient manner.

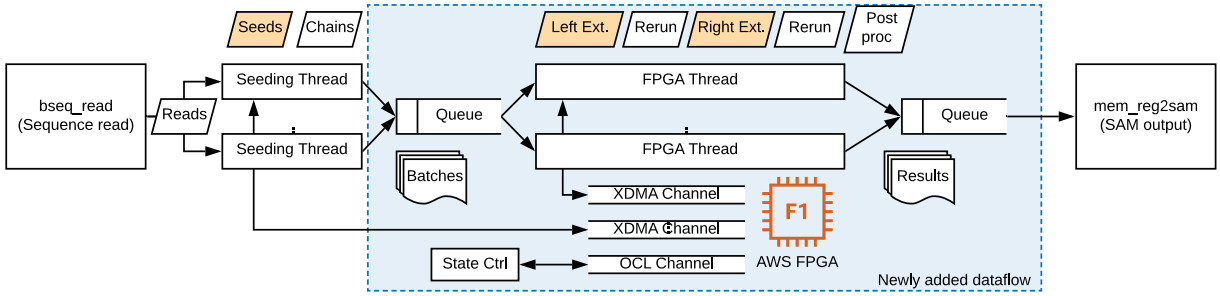

Fig. 12. SeedEx integration with BWA-MEM [6].

B. BWA-MEM Integration

SeedEx FPGA is integrated into the BWA-MEM [6] pipeline. BWA-MEM has three software pipeline stages: sequence read, sequence process, and SAM output stage. The sequence process stage performs the core algorithms, seeding and seed extension, with multi-threading enabled. We isolate the seed extension kernel and restructure it using the FPGA interface. The software workflow is described below and illustrated in Figure 12.

Reads provided by the previous pipeline stage are distributed to, and processed by, seeding threads in the original code block. Seeding threads perform seeding and chaining, and we optionally use the seeding accelerator [35] to accelerate the seeding step. The processed chains are batched and queued for FPGA threads. We employ the producer-consumer model to facilitate load-balancing in FPGA by adjusting the number of FPGA threads.

FPGA threads package the input sequences with metadata and send them to FPGA DRAM over the XDMA channels. 4 XDMA channels are shared by both the FPGA threads and the seeding threads. Multiple threads can concurrently DMA to the FPGA DRAM. After the batch transfer completes, an FPGA thread acquires a lock to control the FPGA state. Once the lock is acquired, it sends the `batch_start` command through the OCL channel to the FPGA's state control register. Then, the FPGA thread continuously monitors the register by the `fpga_peek` interface. Upon receiving the `batch_done` signal, which is established after the writeback completion of the last result entry, it releases the lock for awaiting FPGA threads and retrieves the results over XDMA. When a rerun is required, it is done by the FPGA thread at this juncture.

For a given seed, BWA-MEM performs a left extension (from the left end of the seed) and/or a right extension (from the right end), depending on the seed position in the read. If a right extension is required, after the left extension, the initial score must be updated with the left extension score. This is performed in the middle of parsing left extension results. Multiple FPGA threads interleave to conceal FPGA execution latency.

BWA-MEM dynamically determines whether to perform seed extension for a seed in a chain, based on the preceding extension results for that chain. This decision making is challenging under the batching model and the out-of-order result production of SeedEx. A similar challenge exists in BWA-MEM2 [17], which has a SIMD parallelized seed extension kernel. We take a similar approach to BWA-MEM2; the FPGA processes all seeds in a chain and filters out needless results during the post process rearrangement stage.

After a batch is processed, the result is queued for SAM output. We do not accelerate traceback. This is because, for a seed, only the extension giving the best score needs to be traced back. Thus, tracing back for all extensions is not ideal, especially in a situation with resource limitation.

While our software and hardware implement some customized optimizations specifically designed for BWA-MEM, this is just an example embodiment of SeedEx. The underlying architecture and methods are broadly applicable to different aligners and other applications, as we discuss shortly in Section VII.

TABLE I
BASELINE SYSTEM CONFIGURATIONS.

f1.2xlarge (AWS EC2 instance)	Intel Xeon E5-2686 v4 2.3 GHz; 8/18 vCPUs
L1 I&D cache	8/18 x 32KiB Instruction; 8/18 x 32KiB Data
L2 cache	8/18 x 256KiB
L3 cache	8/18 x 2.5MiB
Memory	122 GiB DRAM
FPGA	Xilinx Ultrascale+ VU9P 64 GiB DDR4; PCIe x16 2.5 M logic elements; 6,800 DSPs

VI. Methodology

Reference Genome and Input Reads: We use the latest major release of the standard human genome assembly (GRCh38/hg38) from the UCSC genome browser [36] for the reference human genome. Index files are generated using BWA-MEM [6]. We store a copy of the 2-bit encoded reference genome on FPGA DRAM to efficiently fetch the encoded reference, bypassing the host machine memory bandwidth constraints. For the input reads, we use real human genome reads with 50× coverage from the Illumina platinum genomes [37] dataset. The dataset consists of the NA12878

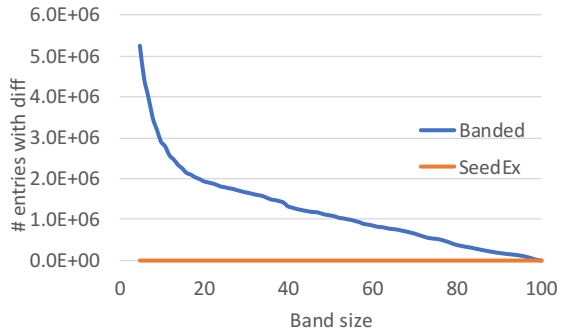

Fig. 13. SeedEx validation results.

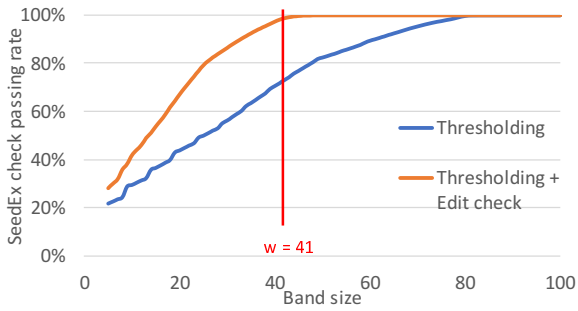

Fig. 14. Passing rate of SeedEx check algorithm.

human reference (single-end ERR194147_1.fastq) consisting of 787,265,109 reads of 101 base pair (bp) length.

System Configuration: SeedEx custom logic is built for and registered as an Amazon FPGA Image (AFI), and loaded to an AWS F1 instance. We use `f1.2xlarge` as a baseline system configuration. The `f1.2xlarge` instance has a 16 nm Xilinx Ultrascale+ VU9P FPGA with 64 GiB DDR4 memory and dedicated PCIe x16 connection. The FPGA is mounted on a system with 8 virtual CPUs (Intel Xeon E5-2686 v4) operating at 2.30 GHz and 122 GiB DDR4 main memory.

SeedEx software interface is integrated with the latest BWA-MEM (release v0.7.17) [6] and BWA-MEM2 (release v2.0pre2) [17] and compared against stock CPU execution. We also compare SeedEx with a GPU aligner (CUSHAW2) [38] on NVIDIA's TITAN Xp. Similar to BWA-MEM's SMEM approach [6], CUSHAW2-GPU identifies maximal-length matches and extends these to form larger gapped alignments. We also compare an ASCI version of SeedEx with GenAx [8] and ERT [35]. We use the default scoring scheme in BWA-MEM for all the aligners.

To study SeedEx's raw alignment throughput, independently of the seeding and software throughput, we compare it against major software and hardware implementations of the seed extension kernel. We use the SeqAn library [39] as the CPU baseline, SW# [40] as the GPU baseline, and Sillax [8] as the hardware baseline. In the following section, we will show a comparison of ASIC implementations.

Synthesis: We synthesize and implement SeedEx using Vivado 2019 on AWS FPGA Development AMI. SeedEx is synthesized with 8 ns clock, while the seeding accelerator uses 4 ns clock. We use an AXI-4 clock bridge to connect the logic designs in two different clock domains. ASIC implementations are synthesized using the Synopsys Design Compiler (DC) in a commercial TSMC 28 nm process.

VII. Results

In this section, we present the seed-extension kernel evaluation of the SeedEx accelerator, followed by application-level evaluation. We compare SeedEx to a full-band Smith-Waterman accelerator (w = 101), and GenAx [8].

A. SeedEx Evaluation

Score validation: We validate the SAM output of all 787,265,109 reads from the SeedEx algorithm *exactly* matches with the one from the native run of BWA-MEM and BWA-MEM2. As a sensitivity study, we measure the expected number of result SAM entries that differ in the baseline output with default setting (band w = 100) and in the output with a particular band size setting. For this study, we sample 10 million random reads and scale the results to the size of the whole genome, for the sake of time. The results are shown in Figure 13. Essentially, this will depict the inaccuracy caused by the BSW heuristic, namely implementing a PE array with a smaller band.

With a small band, the number of different entries is high, being over 5.0×10^6. As the band increases, the number of different entries decreases, eventually reaching 0 when the band size is full. On the other hand, we set the band size for the SeedEx algorithm from 5 to 100, while the optimality checks remain unchanged. The output from the SeedEx algorithm is consistent with the default baseline output regardless of the change in the band size setting.

Passing rates: Figure 14 shows the passing rate for the optimality checks in the SeedEx algorithm. The passing rate for only using thresholding increases as the band increases, but it requires a band size of 70 to reach a passing rate of 95%, and a band size of 81 to reach near-100%. With the addition of the edit-distance check, however, the passing rate can be boosted by 18% on average; for some band setting, the increase of passing rate can be over 30%.

We choose band size of 41 as our configuration, as continuing to increase the band does not significantly benefit the overall passing rate. With a band of 41, the thresholding-only passing rate is 71.76%, and the overall passing rate is 98.19%. Moreover, we notice the passing rate for only using thresholding is slightly over $\frac{2}{3}$, meaning roughly one out of three extensions on average will fail the threshold check and need to run the edit-distance machine. Therefore, we set the ratio of BSW core and Edit machine to 3:1.

Area: Figure 15 shows resource utilization breakdown of a SeedEx FPGA with four SeedEx cores, each with three narrow-band BSW cores and an edit core. Here we can

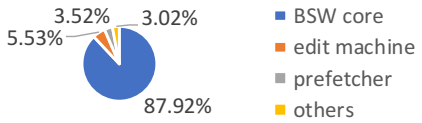

Fig. 15. Resource (LUT) breakdown of SeedEx FPGA.

TABLE II
SEEDING + SEEDEX FPGA RESOURCE UTILIZATION.

Component	Configuration	LUT (%)	BRAM (%)	URAM (%)
Seeding	1 x 6	21.04	10.10	11.81
SeedEx: Controller	1 x 1	0.03	0.01	0.00
SeedEx: I/O Buffers	-	0.49	0.64	0.36
SeedEx: SeedEx Core	1 x 3	12.47	1.14	0.15
SeedEx: Total	-	12.99	1.79	0.51
AWS Interface	-	19.74	12.63	12.20
Total	**-**	**53.77**	**24.52**	**24.52**

observe that a majority of our resources are spent on compute. Prefetching logic and buffering is simplistic and result in little area requirement.

Figure 16 (a) shows the resource utilization of a full-band core (3 BSW cores of $w = 101$) and a SeedEx core. SeedEx core has smaller resource utilization because the BSW core scales proportionally to the band size, as previously shown in Figure 4. The LUT utilization is improved by 2.3×. Although SeedEx has the overhead of edit machines, which accounts for 5.53% of total resources, area reduction due to the small band size more than amortizes the overhead.

Figure 16 (b) compares the resource utilization of BSW core and Edit cores with different optimizations applied. All machines use the same band size of 41. Reduced edit scoring data paths result in 1.82× smaller LUT utilization compared to the baseline BSW core. Delta encoding enables 3-bit arithmetic in PEs and leads to 3.11× reduction. Finally, half-width configuration for the trapezoid-shaped DP matrix results in 6.06× area reduction.

Throughput: FPGA with only SeedEx performs at 43.9M extensions per second (Figure 16 (c)). This configuration consists of 36 narrow-band BSW cores organized in a hierarchy of three clusters, and four SeedEx cores per cluster. The full-band accelerator is comprised of 9 full-band BSW cores. Trying to utilize more area beyond these led to unroutability. Figure 16 shows that SeedEx provides a 6.0× iso-area throughput speed-up. As discussed in Section V, our design achieves nearly optimal throughput. Because of perfect prefetching and appropriate buffering, memory access time is completely hidden. This allows the SeedEx cores to work at near-100% utilization. As a result, throughput scales linearly with the number of clusters.

Note that the latency of seed-extension in SeedEx core is 1.9× lower than the full-band core. This is because buffer initialization of array shift registers and result accumulation time scales proportionally to the band size. Thus, iso-area throughput is improved by 4.4× by the latency and BSW core area, and the rest is attributed to reduced routing complexity.

While the reduced resource utilization of SeedEx is a key

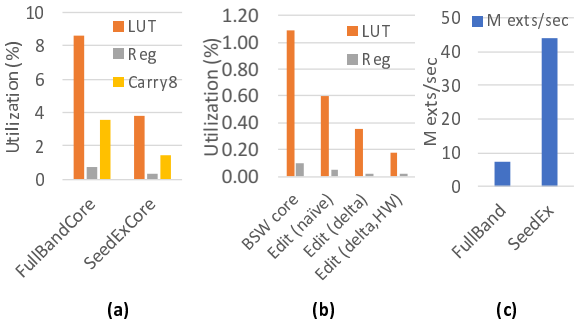

Fig. 16. Area comparison (a) Full band core vs. SeedEx core, (b) BSW core and edit cores. (c) Throughput comparison.

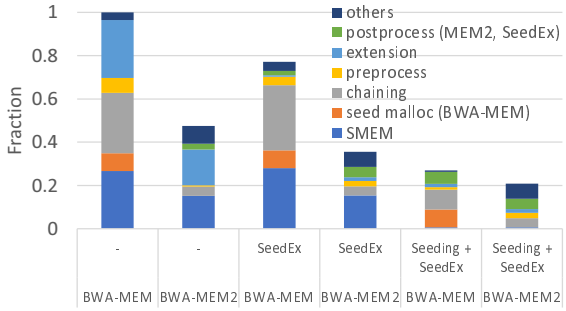

Fig. 17. Normalized end-to-end time breakdown of applications.

contributor to the performance, it has to be noted that SeedEx would need to perform a rerun of some reads. In our setup, about 2% of extensions require a rerun (Figure 13), and we perform it on CPU. We choose the software rerun because it adds negligible performance overhead (Section VII-B), and can be overlapped with FPGA processing across batches. Alternatively, one can use a single full-band machine to perform it, which would add 6% area overhead.

B. Application Evaluation

Table II shows the resource utilization breakdown of a single FPGA image with both the seeding accelerator and SeedEx. We sweep design parameters to match the seeding throughput, and the seed extension throughput, to pick the best configuration that is placable and routable.

Figure 17 compares the overall end-to-end execution time breakdown and throughput of our FPGA accelerated system, hooked up with stock BWA-MEM and BWA-MEM2. In the baseline, seeding (SMEM generation + seed malloc + chaining) and seed extension take a significant fraction ($>$ 85%) of execution time. BWA-MEM2 successfully accelerates the core kernels by a) improving seeding data-structure b) pre-allocating a large contiguous memory region to reduce seed malloc time and facilitate memory prefetching, c) using SIMD parallelization, d) improving cache reuse, and e) implementing code optimizations.

We first note that a software-only implementation of SeedEx (w=5 and full-band reruns on failed cases) results in 14%

speedup for the BSW kernel and 2.8% application speedup in BWA-MEM2 motivating need for a hardware SeedEx accelerator on FPGA. SeedEx accelerator replaces a part of the preprocessing procedure and the seed extension function. By solely using SeedEx for acceleration, it provides 29.6% speedup over BWA-MEM and 33.5% speedup over BWA-MEM2. In BWA-MEM, although SeedEx reduces the execution time of the extension kernel, and makes it invisible in the chart, software seeding becomes a bottleneck. We explore the best thread allocation method, which allocates more than 88% of thread resources to seeding. The remaining threads take turns driving the FPGA and performing downstream processing. The same bottleneck exists in BWA-MEM2. For this reason, we need to accelerate the seeding step to achieve better end-to-end performance.

Combined seeding [35] and seed extension on FPGA leads to Table II. In this design, one SeedEx cluster is paired with our seeding design. Because seeding is DRAM bandwidth limited and requires all four memory channels, this cluster is placed on the memory channel with the least seeding bandwidth requirement. With high FPGA congestion in this combined design, two clock domains were utilized to alleviate place and route (P&R) complexity. A clock of 4ns was used for the DRAM-AXI4 interconnect and seeding, and 8ns clock was used for seed extension. AXI-4 interconnection networks, requiring large data buses, place limits on FPGA P&R capabilities. Sweeping our design parameters to maximize area utilization, with successful P&R, limits our design to around 50-60% LUT utilization on AWS F1's VU9P FPGA.

Figure 17 also shows the overall speedups with both seeding [35] and seed extension accelerators integrated. We observe 3.75× speedup over BWA-MEM and 2.28× speedup over BWA-MEM2. Further, we do not observe a bottleneck in FPGA memory access, nor in PICe communication.

The seeding accelerator [35] improves SMEM generation and part of the chaining procedure. Both Seeding and SeedEx provide 1.5 M reads/s throughput one FPGA instance. While this throughput is significantly larger than the software throughput, overall throughput is bounded by non-accelerated portions. These bottlenecks can be solved by two possible approaches. First, the host can dedicate more threads to the non-accelerated portions. We find that existing F1 instances have severely under-provisioned CPU. The `f1.2xlarge` instance used for our implementation has only 8 vCPUs, thus the host processing is not being able to keep up with SeedEx's high throughput, with underutilized PCIe bandwidth. The discrepancy of software throughput and accelerator throughput can be balanced by a server configuration with more threads, while keeping the same FPGA resources. Likewise, a comparison to an FPGA-less AWS configuration with more cores can potentially undermine SeedEx. If we compare `c5d.9xlarge` (36 vCPUs) with `f1.2xlarge` with 32 vCPUs and SeedEx, however, the FPGA acceleration provides a 1.9× speedup. Note that the vCPUs account for an insignificant cost of the AWS F1 FPGA platforms (price per vCPU $94, FPGA $5500). Therefore, about 2× end-to-end speedup holds true for a CPU+FPGA system with 8 vCPUs, and also 32 vCPUs. Second, SeedEx can be implemented as a part of an end-to-end hardware aligner where all the steps of alignment are done on the FPGA. In our implementation, we only map seeding and seed extension to the FPGA. The remaining workload components become the Amdahl's bottleneck. Of course, end-to-end hardware aligner sacrifices flexibility for performance.

TABLE III
AREA AND POWER OF ASIC SEEDEX.

	Configuration	Area (mm^2)	Power (mW)
I/O buffer	4KiB	0.08	139.5
RAM	2.25KiB x 4	0.31	548.2
BSW cores	12	0.43	288
Edit cores	4	0.04	59.2
Rerun core	1	0.084	35.5
SeedEx Total		0.98	1.10W
ERT	x8	27.78	8.71W
Total		28.76	9.81W

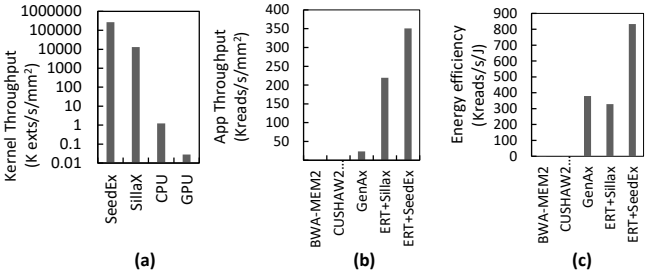

Fig. 18. ASIC SeedEx performance. (a) Extension kernel throughput, (b) Application throughput, (c) Energy efficiency.

C. ASIC Implementation

An ASIC version of a SeedEx design with 12 BSW cores + 4 edit machine cores + 1 full-band rerun core has an area of 0.98 mm^2, power of 1.84 W, and clock period of 0.49 ns. Figure 18 (a) shows the area-normalized seed extension kernel throughput of SeedEx, Sillax, CPU (SeqAn), and GPU (SW#). We observe GPU-based solutions face high synchronization overheads for short reads leading to low performance. Sillax suffers from quadratic scaling of PEs (number of PEs is $\mathcal{O}(K^2)$, where $K = 32$ and band $w = 2K + 1$), while SeedEx's linear scaling and reduced band size provide 20× better performance. The area and power of an ERT-integrated SeedEx ASIC design (scaled for 1.2GHz frequency of ERT) is shown in Table III. Figure 18 (b) and (c) compare area normalized performance and energy efficiency of CPU, GPU, GenAx, ERT + Sillax, and ERT + SeedEx. The default setup of ERT spares 16.08mm^2 (36.5%) of area and 18.48W (62.9%) of power for Sillax. SeedEx reduces area by 16× and power by 10×. This results in 1.56× better overall iso-area performance (b) and 2.45× better energy efficiency (c). When compared GenAx, SeedEx improves iso-area performance by 14.6× and energy efficiency by 2.11×.

D. Discussion

Long Reads: SeedEx has broad applicability by virtue of accelerating read alignment, which is an important precursor step for several downstream studies (tertiary analyses) in next-generation sequencing, such as variant detection (DNA-seq), gene expression profiling (RNA-seq), and protein-DNA interactions mapping (ChIP-Seq).

While this work focused on short-reads, the scalable design of SeedEx can improve secondary analysis for long read technologies ($\geq$ 1Kbp). Unlike the "pure seed-and-extend" strategy of short read aligners (BLAST/BWA-MEM) which could increase w (acceptable edit distance threshold) for long reads, a number of existing long read aligners including BLASR [41] and minimap2 [42] take the "seed-and-chain-then-fill" strategy, allowing small values for w without accuracy loss. For example, minimap2 uses a Needleman-Wunsch based algorithm to perform global alignments between seeds, to filter out internal seeds that can lead to long insertions or deletions. We observe this step takes 16% to 33% of the execution time. SeedEx can be directly applied to this kernel, performing optimal global alignment with a small area.

Other Applications: The proposed SeedEx scheme is broadly applicable to dynamic programming algorithms of which DP calculation has locality in a single dimension (e.g., position in a sequence, time series, etc.). Following is the example applications that can benefit from the SeedEx check approach.

Dynamic Time Warping (DTW) measures similarities between two temporal sequences that may vary in speed. This algorithm has been applied to temporal sequences of video, audio, and graphics data and has been used in speech recognition and pattern matching [43], [44]. While DTW can calculate an optimal match between two given sequences, due to the nature of time-local similarities of two sequences being compared, DTW often uses a fixed time window $[t, t + w]$ varying t [45]. This is conceptually similar to the banded version of the Needleman-Wunsch algorithm. Our proposed scheme is helpful to guarantee optimality even with small time windows that can contribute to reducing hardware resources or processing time.

Longest Common Subsequence (LCS) problem can also be solved with a similar dynamic programming algorithm. It finds the longest common subsequence, and its algorithm is similar to the Smith-Waterman.

VIII. Related Work

In this work, we use Enumerated Radix Trees (ERT) [35] for accelerating seeding. ERT utilizes a radix tree structure to improve data locality and data usage for the SMEM generation algorithm, while generating the same seeds as BWA-MEM [6]. Similar to ERT, BWA-MEM2 [17] also trades off memory capacity for memory bandwidth, but ERT has superior bandwidth efficiency with the tree structure.

Edico Genome's FPGA-based DRAGEN [16] achieves 2.3 M reads/s on an f1.4xlarge instance with two FPGAs, but it is not binary compatible with BWA-MEM. It utilizes hash-based indexing for seeding. This produces a large number of hits and seeds that need to be verified by seed extension and often need to be coupled with filtration techniques.

Darwin [23] is a recent work that demonstrates impressive throughput for long read alignment. Darwin also uses a hash-based lookup for a small number of base pairs and uses a binning mechanism to choose valid seeds from a large number of query hits.

Several FPGA-based hardware accelerators [13]–[16] have been proposed for the Smith-Waterman algorithm. The majority use a systolic array of $\mathcal{O}(N)$ processing elements (PEs). There also exist works that implement a banded Smith-Watermanm algorithm [23]–[25] which have w PEs to compute the band. While our core systolic PE design is minimalistic and has similar components to the prior works, our architecture supports several function knobs to enable BWA-MEM specific optimizations, progressive score initialization and reduction to avoid global wiring, and end-to-end protocol to talk to the software with highly optimized dataflow using prefetching and buffering, in addition to the optimality guaranteed alignment.

GenAx [8] proposes an automata-based Smith-Waterman accelerator. While many prior automata-based works [46]–[50] implement Levenshtein Automata (LA), which is known to be functionality equivalent to Smith-Waterman, they have severe restrictions such as a) only Levenshtein (edit) distance can be used for the scoring scheme, and b) LA is string dependent and every input change requires a prohibitive reprogramming cost. GenAx addresses these issues by introducing string independent local LA (Silla), enabling it to support unique features of sequencing algorithms such as affine gap scoring and clipping. Silla requires $\mathcal{O}(K^2)$ states and has functions equivalent to banded Smith-Waterman.

SeedEx is orthogonal to other implementations of seed extension kernels on hardware. SeedEx offers significant performance improvements by reducing area, compared to full-band machines, and provides the optimality guarantee with little area overhead, compared to narrow-band machines.

IX. Conclusion

As Moore's Law tapers off, hardware acceleration for genomics applications has become essential. Cloud services are starting to make FPGA instances accessible to a broader market, leading to their adoption by most sequencing services. This work presents a novel architecture that uses a speculation-and-test approach for optimal seed extensions with narrow-band accelerators. Our implementation of SeedEx on a cloud-based FPGA achieves 6.0$\times$ iso-area throughput improvement. SeedEx integrated with a seeding accelerator improves the execution time by 2.3$\times$ over a state-of-art BWA-MEM2 software baseline.

Acknowledgment

We thank the anonymous reviewers for their suggestions which helped improve this paper. This work was supported in part by the NSF under the CAREER-1652294 award and the Applications Driving Architectures (ADA) Research Center, a JUMP Center co-sponsored by SRC and DARPA.

References

[1] M. A. Hamburg and F. S. Collins, "The path to personalized medicine," *N Engl J Med*, vol. 2010, no. 363, pp. 301–304, 2010.

[2] "Impact of cancer genomics on precision medicine for the treatment of cancer." [Online]. Available: https://cancergenome.nih.gov/cancergenomics/impact

[3] E. D. Pleasance, R. K. Cheetham, P. J. Stephens, D. J. McBride, S. J. Humphray, C. D. Greenman, I. Varela, M.-L. Lin, G. R. Ordóñez, G. R. Bignell *et al.*, "A comprehensive catalogue of somatic mutations from a human cancer genome," *Nature*, vol. 463, no. 7278, pp. 191–196, 2010.

[4] A. Lacour, A. Espinosa, E. Louwersheimer, S. Heilmann, I. Hernández, S. Wolfsgruber, V. Fernández, H. Wagner, M. Rosende-Roca, A. Mauleón *et al.*, "Genome-wide significant risk factors for alzheimer's disease: role in progression to dementia due to alzheimer's disease among subjects with mild cognitive impairment," *Molecular psychiatry*, vol. 22, no. 1, pp. 153–160, 2017.

[5] Y. Cho, C.-H. Lee, E.-G. Jeong, M.-H. Kim, J. H. Hong, Y. Ko, B. Lee, G. Yun, B. J. Kim, J. Jung *et al.*, "Prevalence of rare genetic variations and their implications in ngs-data interpretation," *Scientific Reports*, vol. 7, no. 1, p. 9810, 2017.

[6] H. Li, "Aligning sequence reads, clone sequences and assembly contigs with bwa-mem," *arXiv preprint arXiv:1303.3997*, 2013.

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

[8] D. Fujiki, A. Subramaniyan, T. Zhang, Y. Zeng, R. Das, D. Blaauw, and S. Narayanasamy, "Genax: A genome sequencing accelerator," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 69–82.

[9] E. Sperling, "How much will that chip cost?" *Semiconductor Engineering*, 2014. [Online]. Available: https://semiengineering.com/how-much-will-that-chip-cost/

[10] M. Farrar, "Striped smith–waterman speeds database searches six times over other simd implementations," *Bioinformatics*, vol. 23, no. 2, pp. 156–161, 2006.

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

[12] C. W. Yu, K. Kwong, K.-H. Lee, and P. H. W. Leong, "A smith-waterman systolic cell," in *International Conference on Field Programmable Logic and Applications*. Springer, 2003, pp. 375–384.

[13] X. Jiang, X. Liu, L. Xu, P. Zhang, and N. Sun, "A reconfigurable accelerator for smith–waterman algorithm," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 54, no. 12, pp. 1077–1081, 2007.

[14] W. Tang, W. Wang, B. Duan, C. Zhang, G. Tan, P. Zhang, and N. Sun, "Accelerating millions of short reads mapping on a heterogeneous architecture with fpga accelerator," in *Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual International Symposium on*. IEEE, 2012, pp. 184–187.

[15] Y.-T. Chen, J. Cong, J. Lei, and P. Wei, "A novel high-throughput acceleration engine for read alignment," in *Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23rd Annual International Symposium on*. IEEE, 2015, pp. 199–202.

[16] R. McMillen and M. Ruehle, "Bioinformatics systems, apparatuses, and methods executed on an integrated circuit processing platform," https://www.google.com/patents/US9014989, Apr. 21 2015, uS Patent 9,014,989.

[17] M. Vasimuddin, S. Misra, H. Li, and S. Aluru, "Efficient architecture-aware acceleration of bwa-mem for multicore systems," in *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2019, pp. 314–324.

[18] M. Šošić and M. Šikić, "Edlib: a c/c++ library for fast, exact sequence alignment using edit distance," *Bioinformatics*, vol. 33, no. 9, pp. 1394–1395, 2017.

[19] M. Zaharia, W. J. Bolosky, K. Curtis, A. Fox, D. Patterson, S. Shenker, I. Stoica, R. M. Karp, and T. Sittler, "Faster and more accurate sequence alignment with snap," *arXiv preprint arXiv:1111.5572*, 2011.

[20] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of molecular biology*, vol. 48, no. 3, pp. 443–453, 1970.

[21] E. W. Myers and W. Miller, "Optimal alignments in linear space," *Bioinformatics*, vol. 4, no. 1, pp. 11–17, 1988.

[22] O. Gotoh, "Optimal sequence alignment allowing for long gaps," *Bulletin of mathematical biology*, vol. 52, no. 3, pp. 359–373, 1990.

[23] Y. Turakhia, G. Bejerano, and W. J. Dally, "Darwin: A genomics co-processor provides up to 15,000x acceleration on long read assembly," *SIGPLAN Not.*, vol. 53, no. 2, p. 199–213, Mar. 2018. [Online]. Available: https://doi.org/10.1145/3296957.3173193

[24] B. Harris, A. C. Jacob, J. M. Lancaster, J. Buhler, and R. D. Chamberlain, "A banded smith-waterman fpga accelerator for mercury blastp," in *2007 International Conference on Field Programmable Logic and Applications*, Aug 2007, pp. 765–769.

[25] P. Chen, C. Wang, X. Li, and X. Zhou, "Hardware acceleration for the banded smith-waterman algorithm with the cycled systolic array," in *2013 International Conference on Field-Programmable Technology (FPT)*, 2013, pp. 480–481.

[26] H. Suzuki and M. Kasahara, "Acceleration of nucleotide semi-global alignment with adaptive banded dynamic programming," *bioRxiv*, 2017. [Online]. Available: https://www.biorxiv.org/content/early/2017/09/07/130633

[27] Y. Liao, Y. Li, N. Chen, and Y. Lu, "Adaptively banded smith-waterman algorithm for long reads and its hardware accelerator," in *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2018, pp. 1–9.

[28] D. S. Hirschberg, "A linear space algorithm for computing maximal common subsequences," *Commun. ACM*, vol. 18, no. 6, pp. 341–343, Jun. 1975.

[29] A. Davidson, "A fast pruning algorithm for optimal sequence alignment," in *Proceedings 2nd Annual IEEE International Symposium on Bioinformatics and Bioengineering (BIBE 2001)*. IEEE, 2001, pp. 49–56.

[30] R. E. Korf and W. Zhang, "Divide-and-conquer frontier search applied to optimal sequence alignment," in *AAAI/IAAI*, 2000, pp. 910–916.

[31] T. Yoshizumi, T. Miura, and T. Ishida, "A* with partial expansion for large branching factor problems." in *AAAI/IAAI*, 2000, pp. 923–929.

[32] R. J. Lipton and D. P. Lopresti, *Using Residue Arithmetic to Simplify VLSI Processor Arrays for Dynamic Programming*. Princeton University, Department of Computer Science, 1986.

[33] R. J. Lipton and D. Lopresti, "A systolic array for rapid string comparison," in *Proceedings of the 1985 Chapel Hill Conference on Very Large Scale Integration*, 1985, pp. 363–376. [Online]. Available: ftp://ftp.cs.princeton.edu/reports/1986/026.pdf

[34] Amazon Web Services. (2020) Amazon ec2 f1 instances. [Online]. Available: https://aws.amazon.com/ec2/instance-types/f1/

[35] A. Subramaniyan, J. Wadden, K. Goliya, N. Ozog, X. Wu, S. Narayanasamy, D. Blaauw, and R. Das, "Accelerating maximal-exact-match seeding with enumerated radix trees," *bioRxiv*, 2020. [Online]. Available: https://www.biorxiv.org/content/early/2020/03/25/2020.03.23.003897

[36] "Uscs genome browser." https://genome.ucsc.edu/.

[37] M. A. Eberle, E. Fritzilas, P. Krusche, M. Källberg, B. L. Moore, M. A. Bekritsky, Z. Iqbal, H.-Y. Chuang, S. J. Humphray, A. L. Halpern *et al.*, "A reference data set of 5.4 million phased human variants validated by genetic inheritance from sequencing a three-generation 17-member pedigree," *Genome research*, vol. 27, no. 1, pp. 157–164, 2017.

[38] Y. Liu and B. Schmidt, "Cushaw2-gpu: empowering faster gapped short-read alignment using gpu computing," *IEEE Design & Test*, vol. 31, no. 1, pp. 31–39, 2014.

[39] A. Döring, D. Weese, T. Rausch, and K. Reinert, "Seqan an efficient, generic c++ library for sequence analysis," *BMC bioinformatics*, vol. 9, no. 1, p. 11, 2008.

[40] M. Korpar and M. Šikić, "Sw#–gpu-enabled exact alignments on genome scale," *Bioinformatics*, vol. 29, no. 19, pp. 2494–2495, 2013.

[41] M. J. Chaisson and G. Tesler, "Mapping single molecule sequencing reads using basic local alignment with successive refinement (blasr): application and theory," *BMC bioinformatics*, vol. 13, no. 1, p. 238, 2012.

[42] H. Li, "Minimap2: pairwise alignment for nucleotide sequences," *Bioinformatics*, vol. 34, no. 18, pp. 3094–3100, 2018.

[43] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 26, no. 1, pp. 43–49, 1978.

[44] R. J. Kate, "Using dynamic time warping distances as features for improved time series classification," *Data Mining and Knowledge Discovery*, vol. 30, no. 2, pp. 283–312, 2016.

[45] Weste, Burr, and Ackland, "Dynamic time warp pattern matching using an integrated multiprocessing array," *IEEE Transactions on Computers*, vol. C-32, no. 8, pp. 731–744, 1983.

[46] P. Dlugosch, D. Brown, P. Glendenning, M. Leventhal, and H. Noyes, "An efficient and scalable semiconductor architecture for parallel automata processing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 12, pp. 3088–3098, 2014.

[47] A. Subramaniyan, J. Wang, E. R. M. Balasubramanian, D. Blaauw, D. Sylvester, and R. Das, "Cache automaton," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-50 '17. New York, NY, USA: ACM, 2017, pp. 259–272.

[48] V. Gogte, A. Kolli, M. J. Cafarella, L. D'Antoni, and T. F. Wenisch, "Hare: Hardware accelerator for regular expressions," in *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*. IEEE, 2016, pp. 1–12.

[49] Y. Fang, T. T. Hoang, M. Becchi, and A. A. Chien, "Fast support for unstructured data processing: The unified automata processor," in *Microarchitecture (MICRO), 2015 48th Annual IEEE/ACM International Symposium on*. IEEE, 2015, pp. 533–545.

[50] Y. Fang, C. Zou, A. J. Elmore, and A. A. Chien, "Udp: A programmable accelerator for extract-transform-load workloads and more," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-50 '17. New York, NY, USA: ACM, 2017, pp. 55–68. [Online]. Available: http://doi.acm.org/10.1145/3123939.3123983