FPGA-based Acceleration for Tracking Audio Effects in Movies

Mihalis Psarakis, Aggelos Pikrakis, Giannis Dendrinos
Department of Informatics, University of Piraeus, Greece
{mpsarak | pikrakis | dendrinos}@unipi.gr

Abstract—In this paper we propose an FPGA-based hardware platform to accelerate an audio tracking method. Our tracking approach is inspired by the problem of molecular sequence alignment and adopts a well-known dynamic programming algorithm (Smith-Waterman algorithm) from the area of bioinformatics. However, the high computational complexity of such algorithms imposes a significant barrier to their adoption by audio tracking systems. To alleviate the time-consuming problem and achieve realistic response times, we propose the acceleration of computationally intensive parts of our tracking method using an FPGA-based platform. Our FPGA accelerator is actually based on the systolization of the Smith-Waterman algorithm proposed in previous approaches for the acceleration of bio-sequence scanning but the special requirements of the audio tracking method impose significant design challenges in the accelerator architecture. The accelerator has been implemented in a Xilinx Virtex-5 device and the experimental results show that it achieves significant speedup compared with the software implementation of the tracking method. The proposed approach has been tested in the context of detecting animal sounds in audio streams from movies, where a basic requirement is to reduce the noisiness of the detection results by means of exploiting the statistical nature of the scores that are generated by the dynamic programming algorithm.

Keywords—FPGA-based acceleration; audio tracking; sequence alignment; Smith-Waterman algorithm

I. INTRODUCTION

The increasing availability of digitally stored movies via numerous distribution channels has highlighted the need for efficient automated annotation mechanisms of the movie content [1]. The problem can be approached from a tracking perspective, i.e., by tracking audio events in the respective audio streams. The spotted events can then serve as metadata for indexing and retrieval operations related to the movie content and can also be exploited by next generation multimedia players to address the requirements of enhanced playback, fast browsing, automated subtitleing and scene description functionality.

To this end, we adopted a query-by-example perspective. The audio events to be tracked are represented by sound effects from commercial databases. In such databases, e.g., the “Sound Ideas - Series 6000 Sound Effects Library”, each sound effect is accompanied by a textual description and similar effects are grouped together to form semantic clusters. Therefore, by tracking such audio effects in a movie, it is possible to associate the respective textual descriptions with the movie content. We have formulated the tracking problem from a sequence alignment perspective and have employed a variant of the well known Smith-Waterman (S-W) algorithm [2]. To align the audio sequences, the S-W algorithm calculates their optimal alignment score using similarity matrices between the movie audio stream and the query effect which are computed based on a similarity function of their precompiled feature vectors.

First, our tracking method applies an offline short-term process in the audio stream of the movie and the target set of the audio effects and extracts their feature vectors based on an MPEG-7 spectral descriptor. Next, it calculates the similarity matrices between the movie segments and the audio effects based on a cosine function of the angle of their feature vectors. Finally, it performs the sequence alignment procedure that consists of repeated runs of the computationally intensive S-W algorithm and is by far the most time consuming phase of our method. Given the large number of patterns to be tracked, the duration of movies and the computationally intensive nature of the adopted sequence alignment algorithm it is therefore imperative to resort to an acceleration platform, which in our case is a FPGA-based solution.

FPGA-based hardware platforms have been previously proposed to accelerate the execution of dynamic programming algorithms (like the Smith-Waterman or the Needleman–Wunsch) for the alignment of bio-sequences [3]-[7]. The structure of these accelerators is typically based on systolic arrays used to parallelize the execution of the alignment process. The fine-grain parallel computing and the hardware reconfigurability enabled by field programmable technology allow the implementation of high-performance and customizable hardware platforms for the execution of such alignment algorithms achieving significant speedups.

In this paper, we propose an FPGA-based platform to accelerate the execution of the sequence alignment procedure of our audio tracking method. The proposed accelerator is also based on the systolization of the S-W algorithm adopted in previous approaches for the acceleration of bio-sequence database scanning but the special requirements of our audio tracking application impose significant design challenges in the systolic array. Although recent approaches taking advantage of the FPGA’s reconfigurability have proposed highly parameterized processing elements tailored to the characteristics of bio-sequences [6] (i.e. various sequence symbol types, sequence sizes and similarity matrices) they can not satisfy the unique requirements of our audio tracking application.
The proposed FPGA-based platform has been implemented in a Xilinx ML501 development board integrating a Virtex-5 XC5VLX50T device. The FPGA device accommodates a systolic array of 100 processing elements for the execution of the S-W algorithm and additional logic for controlling the entire alignment phase. The processing element design is an enhanced version of the circuit design proposed in [3]. The host CPU executes the first steps of the audio tracking method, i.e. the feature extraction and the calculation of the similarity function, and transfers the similarity matrices to the FPGA board through the PCIe interface. The accelerator performs the sequence alignment phase of the method which comprises iterative executions of the S-W algorithm. The profiling of the full software implementation of the proposed audio tracking method determined that the sequence alignment phase occupies more than 85% of the overall execution time. The experimental results demonstrated both the validity of the proposed method and the speedup improvements. The FPGA accelerator achieves up to 127X speedup for a single sequence alignment phase and up to 85X speedup for the entire task of tracking an audio effect in a movie. Due to its computational efficiency, the method made it feasible to extract alignment scores for a sufficiently large number of permutations of the feature sequence per movie segment, thus permitting the implementation of a robust hypothesis testing mechanism that led to a significant reduction of the noisiness of the decision sequence.

The paper is organized as follows. Section 2 describes our audio tracking method and the variant of the S-W algorithm. Section 2 presents previous hardware implementations of the S-W algorithm and highlights the special requirements of our audio alignment application. Section 3 presents the FPGA accelerator architecture. Section 4 analyzes the experimental results while Section 5 concludes the paper.

II. AUDIO TRACKING METHOD

A. Feature Extraction

At a first step, the audio stream of the movie is short-term processed by means of a moving window technique. After extensive experimentation and as a trade-off between computational complexity and detection accuracy, the recommended window length was set to 46.4 ms (equivalently, 1024 samples for a sampling frequency of 22050 Hz), with zero overlap between successive frames.

At each frame, the MPEG-7 Audio Spectrum Envelope (ASE) is extracted [8]. The ASE is a generic, multidimensional spectral descriptor which is based on the idea that adjacent DFT coefficients of the short-term spectral representation of the signal can be grouped (added) to form a single bin, thus producing a simplified version of the spectrum of the signal. The level of spectral refinement can be controlled by means of an appropriate parameter. The ASE is not an optimal feature in any sense, but its generic nature and our experimental evidence suggest that it is clearly a good candidate for inclusion in future research. In our study, the ASE is computed as follows:

- The audio stream is first resampled at $F_s=22050$ Hz and is broken to non-overlapping frames. At each short-term frame, the DFT is computed.
- Taking frequency 1000Hz as a starting point, the spectrum is divided to a set of bands which are logarithmically distributed between 62.5Hz and 11025Hz. The spectral resolution, $r$, of each band is $r = 2^i = 2^2 = 0.25$ octaves and the edges, $loF_b$ and $hiF_b$, of each frequency band are computed as $loF_b = 62.5*2^{(b-1)/r}$ and $hiF_b = 62.5*2^{(b+1)/r}$, where $1 \leq b \leq 32 = (8/r)$. In the above computations, $j$ is the parameter that controls spectral refinement.
- After the bands have been set, the sum of power coefficients in each band yields the respective ASE coefficient. In the end, the DFT coefficients below 62.5Hz are summed to yield one more ASE coefficient.

In our study, due to the fact that the sampling frequency is 22050Hz (after the resampling operation), the actual number bands becomes 31. Therefore, the computation of the ASE yields a 31-dimensional, continuous-valued feature vector per short-term frame.

Let $X_i = \{x_1, x_2, ..., x_M\}$ be the resulting sequence of ASE vectors, where $F$ is the number of short-term frames of the movie. $X_i$ is then augmented with an estimate of its first derivative (delta sequence). This is a common approach in audio processing in order to capture the dependency between adjacent frames. In our case, the first-order derivative, $dx_i$, of the $i$-th feature vector is estimated by the equation $dx_i = G(X_{i+1} - X_{i-1})$, where $G=0.375$ is a gain factor and is included for scaling purposes. The result of this operation is that the feature vector becomes 62-dimensional.

The same type of short-term processing is also applied on the audio pattern to be detected and let $Y = \{y_1, y_2, ..., y_N\}$ be the respective feature sequence, where $N$ is the length of $Y$.

B. Variant of the Smith-Waterman Algorithm

At the next step, $X_i$ is divided into non-overlapping long-term segments ($M$ frames long). At each segment, a variant of the S-W algorithm is employed to detect instances of the audio pattern. Without loss of generality, assume that feature sequence $Y$ and a segment, $X = \{x_1, x_2, ..., x_M\}$, of the feature sequence $X$, are placed on the horizontal and vertical axis of a search grid, respectively. The similarity, $S(i, j)$, between feature vector, $x_i$ (of $X$) and $y_j$ (of $Y$), is defined as follows.

$$S(i, j) = \frac{\sum_{k=1}^L x_i(k) \cdot y_j(k)}{\sqrt{\sum_{k=1}^L x_i^2(k)} \cdot \sqrt{\sum_{k=1}^L y_j^2(k)}}$$

where $L=62$ in our case. $S(i, j)$ is then multiplied by 100 and is rounded to the closest integer, so $S(i, j)$ takes integer values in the range [-100, 100] (100 corresponds to a perfect match).

After $S(i, j)$ has been computed for every pair of feature vectors an iterative (multipass) procedure is started that consists of two stages:

Stage 1: Matrix $S$ is passed to the FPGA accelerator, which returns the alignment score, $m_n$, the coordinates of the corresponding node, $(i_n, j_m)$, the predecessor of each node...
Stage 2: $m_i$ is normalized by dividing it with the perfect score, $N^*100$. If the normalized score exceeds a predefined similarity threshold, $T_r$, we proceed with applying the Wilcoxon rank sum test, in order to check the null hypothesis that the median value of the $r_i$’s is equal to the true (non-normalized) score, $m_i$. If the p-value of the test is less than 0.1, the null hypothesis is rejected, and $m_i$ is considered to correspond to a successful pattern detection, otherwise, $m_i$ is rejected. If $m_i$ is accepted, a backtracking operation follows: we start from node $(i_{ms}, j_{ms})$ and visit the chain of predecessors, until a null node has been reached. If $(i_{i,j})$, $(i_{2,j_2}), \ldots, (i_{ms}, j_{ms})$ is the extracted best path, then the endpoints of the detected pattern lie on frames $i_1$ and $i_{ms}$. In order to detect a second instance of the pattern (and so on), we repeat stages 1 and 2 but we define that the rows of the $S$ matrix that lie between $i_1$ and $i_{ms}$ consist of stop-nodes, i.e., they are no longer allowed to be part of an extracted best path and can only serve as assisting nodes during the S-W processing stage.

III. FPGA IMPLEMENTATION OF THE SMITH-WATERMAN ALGORITHM

In this Section, we first describe the S-W algorithm basics and then present previous approaches proposing hardware implementations of the S-W algorithm for bio-sequence database scanning. Next, we highlight the special features and the design challenges of our accelerator.

A. Smith-Waterman Algorithm Basics

The Smith-Waterman (S-W) algorithm is a dynamic programming algorithm that finds the most similar subsequences (i.e. the best scoring local alignment) of two sequences. The S-W algorithm computes a similarity score that represents the cost to transform one subsequence into another. In this transformation, two elementary operations are allowed: substitution and insertion/deletion (also known as gap operation). The transformation requiring the minimum number of operations to change one sequence into another produces the highest similarity score.

Consider two sequences $X$ and $Y$ of length $M$ and $N$, respectively. The S-W algorithm computes the optimal alignment (or similarity) score $H(i,j)$ of two subsequences $X[0..i]$ and $Y[0..j]$ using the following recurrent equation:

$$H(i,j) = \begin{cases} 0 & \text{if } i = j = 0 \\ E(i,j) & \text{if } i > 0, j > 0 \\ F(i,j) & \text{if } i = 0, j > 0 \end{cases}$$

where $S$ is the substitution cost matrix of the characters consisting the alphabet of the target sequences. In bio-sequence alignment applications, specific substitution cost matrices are adopted obtained from probabilistic models, e.g. the BLOSUM62 which is a 20×20 matrix for aminoacids.

The initial values of the matrix are given by $H(i,0) = E(i,0) = H(0,j) = F(0,j) = 0$. In the above equation, the affine gap penalty is considered: $\alpha$ is the cost of the first gap and $\beta$ is the cost of the following gaps. Some applications use the linear gap penalty where $\alpha = \beta$.

B. Related Work

Previous approaches [3]-[7] for bio-sequence scanning applications have proposed hardware implementations of the S-W algorithm. These hardware implementations map the calculation of the similarity matrix to a linear array of processing elements (PEs). Each PE in the array is initially assigned a character of the query bio-sequence while the subject bio-sequence is shifted systolically through the array of PEs. In each cycle, each PE of the systolic array performs an elementary computation of the S-W algorithm. Thus, the calculation of the similarity matrix for two sequences with length $M$ and $N$ lasts $M+N-1$ cycles instead of $O(M*N)$ cycles required in a sequential processor.

Such a PE design that has been proposed in [3] is shown in Fig. 1. The PE integrates local memory to store the $H(i,j-1), H(i-1,j)$ and $H(i-1,j-1)$ and a LUT (Look-up table) to store the corresponding column of the substitution cost matrix. It computes the similarity score in a two-cycle pipelined fashion: in the first cycle it look-ups the $S(X_i,Y_j)$ cost and appends it to the $H(i-1,j-1)$ while in the second cycle it appends the gap penalties to the locally stored $F(i-1,j)$ and $E(i,j-1)$ to calculate the similarity score $H(i,j)$ which it passes to the next PE. It also monitors the scores flowing through it to pick up the maximum score $\text{Max}(i,j)$ which is stored locally and forwarded to the next PE.
In [3] Oliver et al. proposed the hyper customization of the systolic array to make better use of the fine-grain and runtime reconfiguration capabilities of the contemporary FPGAs. Their FPGA implementation of the S-W algorithm achieves a 170X speedup for linear gap penalties and 125X for affine gap penalties compared with a desktop computing platform. In [4] Jiang et al. proposed modifications to the original PE design of [3] and a zig-zag floorplan approach to improve the performance of the S-W accelerator. In [5] Zhang et al. implemented the S-W algorithm for both DNA and protein sequences scanning in a reconfigurable supercomputing platform. They proposed a multistage PE design with uneven pipeline stage latencies and a compact storage structure for the substitution matrix which reduces significantly the PE area requirements. In [6] Benkrid et al. presented a highly parameterizable FPGA implementation of sequence alignment algorithms (i.e. Smith-Waterman or Needleman-Wunsch algorithms). The proposed FPGA-based skeleton is based on the array of PEs presented in previous approaches which is highly parameterized (i.e. in terms of sequence symbol type, sequence length, gap penalty, etc.) to suit the particular requirements of the user application.

C. Design Challenges of the Audio Sequence Alignment

Although recent approaches taking advantage of the reconfiguration capabilities of the modern FPGA architectures have proposed highly parameterized PEs tailored to the characteristics of bio-sequence scanning applications [6] they can not satisfy the unique requirements of the audio sequence alignment process embedded in our audio tracking application. Next, we analyze the main differences between a bio-sequence alignment process and an audio sequence alignment process and highlight the related design challenges of our FPGA implementation.

Bio-sequence symbols vs. audio spectral descriptors. The bio-sequences are strings of symbols drawn from discrete alphabets (e.g., the alphabet of proteins) and hence, the related similarity function is restricted to yield discrete values. In our application, each audio segment is represented by a sequence of spectral descriptors which take on continuous values in a bounded range. The similarity function also yields continuous values of variable resolution in a bounded range. Thus, a challenging issue is to determine the number representation of the similarity scores in the FPGA design that achieves the best tradeoff between complexity and precision requirements.

Fixed vs. reprogrammable lookup tables. Bioinformatics scientists adopt a specific substitution cost matrix (i.e. the BLOSUM62 matrix) for their alignment algorithm. Thus, in the related FPGA accelerators, where the cost matrix is stored locally in lookup tables of the PEs, the lookup tables are initially configured to hold the entire matrix or a part of it. To facilitate the use of more than one cost matrices [4] or to allow the loading of the query sequence at the start of each scan [3], [6], previous works propose the use of distributed RAMs and in-circuit loading of the cost matrix. However, in any case the contents of the local lookup tables are fixed throughout each database scan. In our application, the audio tracking algorithm calculates a new cost matrix for every pair of audio segments and thus the lookup tables must be reprogrammed for the alignment of every audio effect with every movie segment. This means that the cost matrix reloading time consists a significant part of the sequence alignment time and thus the integration of a fast configuration chain into the systolic array is another design challenge.

Permutation of the audio effect. Our audio tracking method requires multiple executions of the S-W local alignment algorithm for a single audio effect with permuted frames (i.e. in each execution the columns of the similarity matrix are shuffled) in order to apply the Wilcoxon rank sum test. To enable this capability without reloading each time a new matrix to the FPGA device, we must add extra logic for generating in-circuit random permutations of the audio effect frames.

Inactive processing elements. Our audio tracking method proposes a multipass procedure. In each pass it removes from the target movie segment a set of consecutive frames which in the previous pass have been defined as stop nodes. In this case, the corresponding PEs of the systolic array does not contribute to the local alignment score but only passes through the accumulated costs from the previous PEs. To enable this, we must extend the functionality of the PE to support an inactive state and add interface logic to reconfigure the array.

IV. PROPOSED FPGA ACCELERATOR ARCHITECTURE

In our proposed FPGA accelerator architecture we also map the S-W algorithm to a systolic array of PEs. Let’s assume the alignment of two audio segments, a movie segment X and an audio effect Y consisting of M and N frames, respectively. Each PE in the array is assigned a frame of the movie segment X and the audio effect Y is shifted systolically through the array as shown in Fig. 2. The systolic array calculates the similarity matrix H(i,j) of the two audio segments based on the elementary computation of the S-W algorithm. Actually, each PE stores a row of the similarity matrix S, as shown in Fig. 2, which has been precomputed in the host machine, i.e. the PE k stores the values S(Xk,Yj), 1 ≤ j ≤ N which are the substitution costs between the frame Xi and the frames Y1 to YN. Thus, a new similarity matrix is loaded in the systolic array for every new pair of audio segments X and Y while during the alignment process the frame indices of the audio segment Y are shifted through the systolic array in an ascending order.

Figure 2. Proposed systolic array.
We adopt the typical PE design shown in Fig. 1 [3] which we extend to support the extra functionality required in our tracking method. The design of the proposed PE is shown in Fig. 3. The original PE presented as black box in Fig. 3 implements the PE circuit shown in Fig. 1. Affine gap penalty is considered for the PE design with the gap penalties given by \(\alpha = 133\) and \(\beta = 33\).

![Figure 3. Proposed PE design.](image)

The original PE calculates the similarity score \(H(i,j)\) and the maximum score \(\text{Max}(i,j)\) as described in Section III.B. The width of signal \(S) (sw)\) and signals \(E, F, H\) and \(\text{Max} (dw)\) of the S-W algorithm determines the range and the precision of the similarity scores and thus affects our tracking method accuracy. The similarity scores \(S(i,j)\) of feature vectors computed in the first step of our method take values in the range \([-1, 1]\) where 1 corresponds to a perfect match. Since a two decimal point precision is fairly adequate for our application (experiments showed that higher precision does not improve the efficiency of the method), the similarity scores \(S(i,j)\) are rounded to two decimal places and multiplied by 100. Therefore the similarity scores \(S(i,j)\) downloaded to the FPGA accelerator take values in the range \([-100, 100]\); this integer notation of simplifies the arithmetic operations of the S-W algorithm. The range of the similarity scores \(H(i,j)\) depends on the range of \(S(i,j)\) and the number of frames of \(Y(N)\). Assuming \(N\) consecutive frame matches, the maximum possible similarity score is \(N^2 \times 100\).

Three signals have been added to the original PE: \(\text{reset}, \text{freeze}\) and \(\text{stop}\). The \(\text{reset}\) signal clears the locally stored \(E, F\) and \(H\) values while the \(\text{freeze}\) signal stalls the PE operation. When the \(\text{stop}\) signal is active the PE is inserted in an inactive state, which means that it does not perform the substitution operation but only the gap operations (i.e. \(H(i,j) = \text{max}[0, E(i,j), F(i,j)]\)). The PE unit also computes the predecessor of each score (i.e. the direction from which it has been derived) as described in [6] and stores it (a 2-bit code) along with the similarity score.

Two RAMs are used in every PE to store the precomputed similarity matrix \(S\) and the similarity matrix \(H\) computed by the S-W algorithm. The \(S\)-RAMs are loaded via a configuration chain that shifts the values of the matrix \(S\) into the PEs. During the execution of the S-W algorithm, the original PE unit reads the \(S\)-RAM and writes the accumulated score to the \(H\)-RAM. At the end, the contents of the \(H\)-RAMs are shifted out via the same configuration chain. The configuration chain consists of the \(Y\) and \(D\) signals which carry addresses and data, respectively. The width of \(Y (nw)\) depends on the maximum length of the audio effect \(N\) and determines the RAMs depth. Note that the length of the audio effect is programmable through a PCIe-memory mapped register.

The control logic coordinates the operation of the PE and generates the PE control signals and the RAM write signals. The PE has six modes of operation which are shifted systolically to the array through the \(\text{mode}\) signal: (a) \(\text{reset}\): activates the \(\text{reset}\) signal that clears the PE, (b) \(\text{wr}\text{ram}\): writes the \(S\)-RAM via the configuration chain, (c) \(\text{process}\): performs the elementary operation of the S-W algorithm, (d) \(\text{rd}\text{ram}\): reads the \(H\)-RAM via the configuration chain and (e) \(\text{stop1}\) and \(\text{stop2}\): these two modes are used to determine the stop nodes of the array for the multi-pass alignment procedure described in Section II.B. The configuration of the stop nodes requires two cycles: during the first cycle the PE is entered in mode \(\text{stop1}\) and the \(\text{index}\) signal carries the leftmost stop node while during the second cycle (mode \(\text{stop2}\)) the rightmost stop node is indexed. When a PE belongs to a range of stop nodes it activates the \(\text{stop}\) signal. Note that the configuration of the stop nodes can be done accumulatively, i.e. multiple stop ranges can be defined. The \(\text{index}\) signal is also used to index the PEs during the \(\text{wr}\text{ram}\) and \(\text{rd}\text{ram}\) modes. The width of the \(\text{index}\) signal \((\text{mv})\) depends on the number of the PEs and determines the maximum supported length of a movie audio segment \((M)\).

The basic structure of the proposed FPGA-based accelerator is shown in Fig. 4. The FIFOs are used to store the similarity scores and the outcomes of the random permutations. The state machine and a set of PCIe memory mapped control resisters govern the alignment process.

![Figure 4. Proposed FPGA-based accelerator.](image)

The entire alignment process between a movie segment and an audio effect consists of the following steps:
Step1: Download the S matrix. The host machine downloads the S matrix through the PCIe to the downstream (DN) FIFO.

Step2: Write the S-RAMs. The state machine is entered in the wr_ram state and the contents of the DN FIFO are shifted via the configuration chain and stored to the S-RAMs. One counter (0…M-1) is used to index the PEs (index) while a second counter (0…N-1) generates the addresses for the S-RAMs (signal J). This step lasts M+N cycles.

Step3: S-W process. The state machine is entered in the process state and the S-W alignment process is executed by the systolic array. The second counter generates the indices of the audio effect frames in an ascending order which are shifted to the systolic array. The similarity scores and the predecessors are stored in the H-RAM. This step lasts M+N-1 cycles.

Step4: Read the H-RAMs. The state machine is entered in the rd_ram state and the contents of the H-RAMs are shifted out through the configuration chain and stored to the upstream (UP) FIFO. The counters are used to index the PEs and the S-RAMs similar to Step2. This step lasts M+N cycles too.

Step5: Upload the H matrix from the UP FIFO through the PCIe interface. The DN and UP FIFOs are implemented within the FPGA device and their size is M×N×sw-bit and M×N×(dw+2)-bit, respectively.

Step6: Permutation. The S-W algorithm is executed K times with permutated data (i.e. in each execution the frames of the audio effect are shuffled). In each iteration a hardware permutator is used to generate the indices of the frames in a shuffled order. The number of permutations (K) is programmable through a PCIe memory mapped register. The permutator design is described at the end of this section. Note that our tracking method does not need to evaluate the entire similarity matrix for each permutation but only the maximum matching score. The maximum similarity scores of the permutations are stored in the PERM FIFO which size is K×dw-bit. The execution time of the permutation step varies (each iteration lasts more than the M+N-1 cycles of Step3) because the permutator may freeze the operation of the systolic array as described below.

Step7: Upload the permutation data. The host uploads the results of the random permutations from the PERM FIFO through the PCIe interface.

Step8: S-W process with stop nodes: The host determines a range of stop nodes to the accelerator via two PCIe memory-mapped registers and initiates the alignment process once again (only steps 3 to 5 are executed). The configuration of the stop nodes is done in a two-cycle phase that shifts to the array the leftmost and rightmost stop nodes.

The permutator is based on a version of the well-known Fisher-Yates shuffle algorithm [9] and its operation is described by the following pseudocode:

for i from N-1 downto 1 do
  p = floor(log2(i))
  j ← LFSR(0.51) mod 2^p
  while (j > i) do
    j ← LFSR(0.51) mod 2^p
  exchange Y[j] and Y[i]

Assume an audio segment Y consisting of N frames. In each iteration, the permutator picks a random frame Y[j], 0≤j<N and swaps it with the last frame not yet chosen. To generate a random number in every range from 0..N-1 down to 0..1 we use an 52-bit LFSR that provides random numbers and then we apply the modulo operator (where the second operand is always a power of 2). In the case that the result is higher than i then we discard it and try again until the LFSR generates a number in the suitable range. The simulations showed that the mean number of retries for the generation of a valid random number is about 0.5. When the permutator generates an invalid frame number, the control logic stalls the systolic array activating the freeze signal that is applied concurrently to all the PEs.

V. EXPERIMENTAL RESULTS

A. FPGA Accelerator

We implemented the proposed accelerator in a Xilinx Virtex-5 XC5VLX50T device (the Xilinx ML505 development board was used). An integrated PCIe core was used for interfacing with the host machine. The XC5VLX50T device accommodates 100 PEs with 128-entries RAMs after an unconstrained placement. This means that our experimental setup supports the alignment between a movie segment with a maximum length of 100 frames (i.e. 4.64sec assuming a frame length of 46.4ms or 1024 samples at 22.05KHz sampling frequency) and an audio effect with maximum 128 frames (i.e. 5.94sec). These dimensions determine the following circuit parameters: mw=7, nw=7, sw=8, dw=14. Table I compares the area of the proposed PE circuit and the original PE circuit shown in Fig. 1 [3]. The critical path was not affected by the modifications in the PE and thus no performance degradation was observed (max. frequency for both PEs is 114MHz).

<table>
<thead>
<tr>
<th></th>
<th>Slice Registers</th>
<th>Logic LUTs</th>
<th>Memory LUTs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original PE</td>
<td>48</td>
<td>161</td>
<td>16</td>
</tr>
<tr>
<td>Proposed PE</td>
<td>60</td>
<td>201</td>
<td>48</td>
</tr>
</tbody>
</table>

B. Speedup of the Audio Tracking Process

The proposed audio tracking process consists of five major tasks: feature extraction, calculation of the similarity matrix S, sequence alignment based on the S-W algorithm, Wilcoxon rank sum test and backtracking operation. Since the method relies on audio effects included in commercial databases, they can be precompiled and, therefore, the feature extraction task can be performed offline. Software profiling of the program that implements the proposed method, using the Gprof tool, collected the following statistics: similarity matrix calculation 12% (for reading the feature files and calculating the matrix S), S-W algorithm 86%, Wilcoxon rank sum test 1% and backtracking operation 1% (the profiling was done for 90-frames pattern length and 1000 permutations). The profiling statistics showed that accelerating the S-W algorithm using an FPGA platform was an obvious choice to achieve overall speedup for our audio tracking process.
We performed a complete set of experiments to calculate the speedup resulting from the use of the FPGA accelerator. Our experiments include the tracking of five audio effects with different lengths ranging from 0.9sec to 5.5sec ($N = 20, 30, 60, 90$ and 120 frames) in a short movie window (window duration is about 13min and it is divided in 168 segments of 100 frames each) for different number of permutations ($K = 1000, 2000, 3000, 4000, 5000, 6000$ and $8000$). Table II presents three different execution times: (a) $T_{H/W}$: execution time of running a single alignment cycle between a movie segment and an audio segment (i.e. steps 2 to 4 plus step 6) in FPGA, (b) $T_{H/W+IO}$: $T_{H/W}$ plus the time required for the data exchange between the host PC and the board (i.e. plus steps 1, 5 and 7) through the PCIe interface and (c) $T_{S/W}$: execution time of running a single alignment cycle (i.e. steps 2 to 4 plus step 6) in software, i.e. without using the FPGA accelerator. The execution times of Table II have been calculated by averaging the execution times for the alignment between the five audio effects (with different length) and the 168 audio segments of the short movie. The software program of the audio tracking method including the S-W algorithm was implemented in C and compiled using the gcc compiler with -O3 optimization level. The experiments were executed in an Intel Core2Duo E8400 workstation running at 3.6GHz with 4GB DDR2.

<table>
<thead>
<tr>
<th>$K$</th>
<th>$T_{H/W}$ (ms)</th>
<th>$T_{H/W+IO}$ (ms)</th>
<th>$T_{S/W}$ (ms)</th>
<th>Speedup</th>
<th>$T_{S/W}/T_{H/W}$</th>
<th>$T_{S/W}/T_{H/W+IO}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>1.7</td>
<td>6.7</td>
<td>174.2</td>
<td>104</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>2000</td>
<td>3.1</td>
<td>7.0</td>
<td>343.7</td>
<td>110</td>
<td>49</td>
<td>49</td>
</tr>
<tr>
<td>3000</td>
<td>4.6</td>
<td>8.5</td>
<td>551.5</td>
<td>120</td>
<td>65</td>
<td>65</td>
</tr>
<tr>
<td>4000</td>
<td>6.0</td>
<td>8.8</td>
<td>744.3</td>
<td>123</td>
<td>85</td>
<td>85</td>
</tr>
<tr>
<td>5000</td>
<td>7.5</td>
<td>11.1</td>
<td>942.2</td>
<td>126</td>
<td>85</td>
<td>85</td>
</tr>
<tr>
<td>6000</td>
<td>9.0</td>
<td>13.1</td>
<td>1127.8</td>
<td>126</td>
<td>86</td>
<td>86</td>
</tr>
<tr>
<td>8000</td>
<td>11.9</td>
<td>17.2</td>
<td>1501.0</td>
<td>127</td>
<td>87</td>
<td>87</td>
</tr>
<tr>
<td>Average</td>
<td>6.3</td>
<td>10.3</td>
<td>764.3</td>
<td>122</td>
<td>74</td>
<td>74</td>
</tr>
</tbody>
</table>

Timing parameter $T_{H/W}$ includes the execution time of the S-W process plus the time to write and read the RAMs of the systolic array through the configuration chain. As the number of permutations increases, the time to configure the array becomes negligible compared with the total execution time, and therefore the speedup improves up to 122X. In the case of timing parameter $T_{H/W+IO}$, the data transfer time through the PCIe interface occupies a large portion of the total execution time adversely affecting the speedup improvements. Below 2K permutations, the speedup is considerably low since the data transfer time dominates the total S-W process time, whereas above 4K permutations, where the ratio between the transfer time and the S-W execution time is almost constant the speedup saturates at a maximum of 87X. Note that the current implementation does not support DMA – which is under construction – and will further improve the overall speedup.

Fig. 5 presents the overall speedup compared with the software approach for tracking an audio effect in the 13-min movie window. It presents five speedup curves (for different-size audio effects) showing the scaling of the speedup with increasing pattern length and number of permutations. Given the dimensions of the current experimental setup (i.e. the 100 PEs can support movie segments up to 100 frames long) increasing the pattern length more than 120 frames is not meaningfulness. However, migrating the accelerator to a larger FPGA device, able to accommodate a larger systolic array, will allow the processing of longer audio segments which will reveal even higher speedup improvements.

C. Experimental Setup

Our experimental setup involves tracking horse-related sounds in a corpus of the first four episodes of the classic ITV production of the “Adventures of Sherlock Holmes”. Each episode is approximately one hour long. The respective sound effects are: horse trot (13 instances), walk (5 instances), gallop (16 instances) and cart pulling (5 instances). All effects were selected from the “Sound Ideas - Series 6000 Sound Effects Library”. In all experiments, the similarity scores have been normalized to unity (Section II.B).

Fig. 6 presents the percentage of surviving movie segments that yield a similarity score higher than a varying similarity threshold, for the case of “Gallop” sounds, for varying pattern length and without employing any hypothesis testing to filter out results. All three curves exhibit a similar behavior, i.e., once the curve drops below 1%, it decays rapidly. For example, for a pattern length equal to 60 frames and threshold equal to 0.34, only 7 segments survive on the average per movie. In addition, there is no guarantee that a movie segment with a high similarity score can be treated as a successful detection, unless it passes the hypothesis test. This implies that high threshold values are likely to provide very limited recall and that it is better to select a smaller threshold value and resort to a filtering mechanism to reduce the number of results. This type of curve is representative for all types of audio effects that we studied. It has to be noted that we have focused on patterns that are at least 60 frames long, because this pattern size is appropriate for the audio effects that we tested, that consist of prolonged repetitive sounds. If we were interested in shorter sounds, e.g., dog barks, shorter patterns might be desirable.
The required filtering can be achieved by exploiting the similarity scores that have been generated by the random permutations at each movie segment. To this end, we apply the Wilcoxon rank sum null hypothesis test and Fig. 7 presents the resulting curves for the case of Gallop sounds (for the sake of comparison). It can be seen that, on the average, 2 out of 3 segments are filtered out, irrespective of pattern length and similarity threshold. For example, for pattern length equal to 60 frames and similarity threshold equal to 0.21, 35 out of 95 patterns survive. This is important because it permits to relax the need for a high similarity threshold and therefore increase detection recall.

Figure 6. Percentage of surviving segments vs. similarity threshold.

Figure 7. Percentage of surviving segments vs. similarity threshold, after the hypothesis test has been included.

In terms of precision and recall of detected events: (a) If the threshold, \( T_g \), that refers to the normalized pattern similarity (Section II.B), is set to a high value, i.e., 0.3 and above, the precision of detected patterns exceeds 90% (average per movie) if the null hypothesis test is included in the computations. The price to pay is that the respective average recall is relatively low (around 30%), which means that only 3 out of 10 patterns are detected. (b) On the other hand, if \( T_g \) is reduced to a value close to 0.2, average recall increases to 60% and average precision drops to 70% after the returned patterns have been filtered out by the null hypothesis test.

In the current set of experiments, the size of the audio effect is basically more than half the size of the movie segment, due to the nature of sounds under study. Therefore, it is not necessary to scan the movie segment repeatedly, i.e., exploit the technique of stop nodes. This technique has been included for future use, as it is the case with shorter sounds, e.g., dog barks, where the use of stop nodes is expected to reveal instances of repetition in the movie segment.

VI. CONCLUSION

We proposed an FPGA-based hardware platform to accelerate the time consuming sequence alignment phase of an audio tracking method. The experimental results showed that the proposed FPGA platform achieves significant speedup. We believe that the FPGA-based acceleration will facilitate the future adoption of dynamic programming algorithms – which are well established in bioinformatics for bio-sequence scanning applications – in the audio processing domain to build efficient tracking machines. Given that the audio tracking mechanisms may serve in the future as a core technology in more complex, sophisticated systems, for example web-servers for rapid annotation of movies content or next-generation multimedia players with enhanced features (e.g. real time automated subtitling based on tracking audio effects), such FPGA-based platforms will gain increasing importance over the coming years.

REFERENCES