Journal of Supercomputing manuscript No.
(will be inserted by the editor)

Parallel Online Time Warping for Real-Time Audio-to-Score Alignment in Multi-core Systems

Pedro Alonso · Raquel Cortina · F.J. Rodríguez-Serrano · P. Vera-Candeas M. Alonso-González · José Ranilla

Received: date / Accepted: date

Abstract The Audio-to-Score framework consists of two separate stages: *pre-processing* and *alignment*. The *alignment* is commonly solved through offline Dynamic Time Warping (DTW), which is a method to find the path over the distortion matrix with the minimum cost to determine the relation between the performance and the musical score times. In this work we propose a parallel online DTW solution based on a client-server architecture. The current version of the application has been implemented for multi-core architectures (x86, x64 and ARM), thus covering either powerful systems or mobile devices. An extensive experimentation has been conducted in order to validate the software. The experiments also show that our framework allows to achieve a good score alignment within the real-time window by using parallel computing techniques.

Keywords Audio-to-Score Alignment · Dynamic Time Warping (DTW) · Score Following · Parallel Computing · Xeon Phi · ARM

Pedro Alonso
Depto. de Sistemas Informáticos y Computación
Universitat Politècnica de València, Spain
E-mail: palonso@upv.es

Raquel Cortina · M. Alonso-González · José Ranilla
Depto. de Informática
Universidad de Oviedo, Spain
Raquel Cortina E-mail: raquel@uniovi.es
M. Alonso-González E-mail: monica300876@gmail.com
José Ranilla E-mail: ranilla@uniovi.es

F.J. Rodríguez-Serrano · P. Vera-Candeas
Telecommunication Engineering Department
Universidad de Jaén, Spain
F.J. Rodríguez-Serrano E-mail: fjrodriguezserrano@gmail.com
P. Vera-Candeas E-mail: pvera@ujaen.es

1 Introduction

Audio-to-score alignment (or *score matching*) is the task of synchronizing an audio recording of a musical piece with the corresponding symbolic score. There exist two approaches to tackle this problem, which are often called "offline" and "online" alignment. In offline alignment, the whole performance is accessible for the alignment process, i.e. it allows to "look into the future" while establishing the matching. Therefore, this lets to develop non causal algorithms that can reach higher matching precision [1]. This is interesting for applications that do not require the real-time property such as Query-by-Humming [2], intelligent audio editors [3], and as a front-end for many Music Information Retrieval (MIR) systems. Online alignment, also known as *score following*, processes the data in real-time as the signal is acquired. This tracking is very useful for applications such as automatic page turning, automated computer accompaniment of a live soloist, synchronization of live sound processing algorithms for instrumental electro-acoustic composition or the control of visual effects synchronized with the music (e.g. stage lights or opera supertitles).

Audio-to-score alignment is traditionally performed in two steps: feature extraction and alignment. On the one hand, the features extracted from the audio signal characterize some specific information about the musical content. On the other hand, the alignment is performed by finding the best match between the feature sequence and the score, which is where the main efforts of this work are directed on. In fact, classical offline systems rely on cost measures between events in the score and in the performance. Two methods well known in speech recognition have been extensively used in the literature: statistical approaches based on Hidden Markov Models (HMMs) [4] and Dynamic Time Warping (DTW) [5]. HMM based approaches can hardly evaluate the whole range of next state possibilities, in [4] the score position HMM can only hop over a sorted position vector. Classic DTW evaluate all the possible paths over the cost matrix, besides, our online approach is able to evaluate all the score position options at each acquired sound signal (so-called frame). Moreover, if a multi-layer HMM were implemented, it would need a training process for setting up all each hop likelihood. On the other hand, each DTW hop cost can be measured with a cost function that compares one state with all the others.

Although these techniques achieve their best performances in their offline version, there are so many applications that require the use of an online version of the algorithm. For example, automatic accompaniment [7], real-time sound source separation[8], automatic page turning [9] or score following [10]. All these applications, apart from requiring an online implementation, are suitable for mobile devices. Moreover, the Score Alignment system must give an output on time (i.e. the latency is limited) and this latency limitation is usually set around tens of milliseconds. For these reasons, we propose a parallel implementation for the DTW solution, so that the Score Following system could be run efficiently, not only over high capacity computers, but also over low performance devices, e.g. like those based on ARM processors.

In this paper we propose an online score following framework that yields to a server-client model application. Given the solution for the first stage (feature extraction) [11], this work focusses on the second stage: real-time audio-to-score alignment, which is in turn performed in two steps. First, the matching measure between events in the score and in the performance is defined. In particular, a cost matrix is estimated using a fast signal decomposition method previously developed in [12] that uses the spectral patterns fixed from the previous stage. Second, the DTW method is applied.

The paper shows an extensive study of the behaviour of the application running on x86/x64 architecture processors, on coprocessors like the Intel® Xeon Phi™, and on processors that implement the ARM architecture. With the last type of devices, we aim at a wide range of mobile devices accounting for tablets and smartphones. In all cases we use the sequential (one core) and the parallel version implemented in OpenMP for all the cores available in the architecture. The experiments have been carried out under both the Linux and the Android operating systems. This large set of combinations of target machines under test and operating systems allows us to validate our system framework as a useful tool for solving the online score following problem.

In the next section we introduce the concept of audio-to-score alignment and briefly the background around this topic. Section 3 presents the software architecture system and analyses the computational aspects of the application from a theoretical point of view. The experiments are all shown in Section 4. The paper is closed with a conclusions section.

2 DTW for Score Following background

Early works in score following were performed mainly using string matching techniques. Later, with the advent of faster computers, different approaches were developed using techniques such as DTW, HMM, hybrid graphical models [13], neural networks [14], or a conditional random field model [1].

In this paper we will use a low complexity signal decomposition method that obtains a distortion matrix for each combination of notes per frame. This matrix is directly used by a DTW algorithm to obtain the optimum path and thus perform the alignment.

Dynamic Time Warping (DTW) is a technique for aligning time series or sequences which has been intensively applied by the speech recognition community [15] and used in many fields. The series are represented by 2 vectors of features $\mathbf{U} = \{u_1, ..., u_n, ...u_N\}$ and $\mathbf{V} = \{v_1, ..., v_m, ..., v_M\}$ where n and m are the point indices in the time series. Letters N and M represent the length of time series $\mathbf{U}$ and $\mathbf{V}$, respectively. As a dynamic programming technique, it divides the problem into several sub-problems, each of which contributes in calculating the distance (or cost function) cumulatively.

The first stage in the DTW algorithm consists of filling a local distance matrix (a.k.a cost matrix) D as follows:

$$D(n, m) = \psi(u_n, v_m) ,$$

where matrix D has $N \times M$ elements, each one representing the match cost between every two points in the time series. The cost function ψ could be any cost function that returns 0 for a perfect match, and a positive value otherwise (e.g. euclidean distance).

In the second stage (*forward step*), a warping matrix C is filled recursively as:

$$C(n,m) = \min \left\{ \begin{array}{c} C(n, m - c_m) + D(n,m) \\ C(n - c_n, m) + D(n,m) \\ C(n - c_n, m - c_m) + \sigma D(n,m) \end{array} \right\} , \quad (1)$$

where c_n and c_m are the step size at each dimension and the range from 1 to α_n and 1 to α_m, respectively. Scalars α_n and α_m are the maximum step size at each dimension. Parameter σ controls the bias toward diagonal steps. Entry $C(n,m)$ is the cost of the minimum cost path from $(1,1)$ to (n,m), and $C(1,1) = D(1,1)$.

Finally, in the last stage (*traceback step*), the minimum cost path

$$\mathbf{w} = \{w_1, \ldots, w_k, \ldots, w_K\} \ , \quad (2)$$

is obtained by tracing the recursion backwards from $C(N,M)$. Each w_k is an ordered pair (n_k, m_k) such that $(n_k, m_k) \in w$ means that the points u_n and v_m are aligned. The cost of a path $C(w)$ is the sum of the local match costs of the path:

$$C(w) = \sum_{k=1}^{K} C(n_k, m_k) \ .$$

Therefore, the goal of the DTW algorithm is to find a minimum cost path $\mathbf{w}$ (2) which, in addition, satisfies the following three conditions:

1. Boundary condition: $w_1 = (1,1)$ and $w_K = (N,M)$.
2. Monotonicity condition: $n_{k+1} \geq n_k$ for all $k \in [1, N-1]$, and $m_{k+1} \geq m_k$ for all $k \in [1, M-1]$.
3. Step size condition: $n_{k+1} \leq n_k + 1$ for all $k \in [1, N-1]$, and $m_{k+1} \leq m_k + 1$ for all $k \in [1, M-1]$.

This approach has the advantage of being computationally simple and can be applied to audio-to-score synchronization. One of the first works that applied DTW to music alignment was presented in [6] following the standard definition of DTW and using "Peak Structure Distance" (PSD). Another possibility is to use discrete chromagrams of the audio signal as features for both sequences as shown in [16]. Some approaches were proposed to reduce the complexity in time and space, such as [17–19]. Adaptive approaches have also been proposed to overcome structure changes by allowing partial synchronization path searches in the DTW alignment [20] or in the online context, by running several trackers in parallel [21]. Finally, Dixon [5] proposes an online DTW algorithm to follow piano performances, where each audio frame is represented by an 84-d vector, corresponding to the half-wave rectified first-order difference of 84 spectral bands. This onset-informed low-level feature works well for

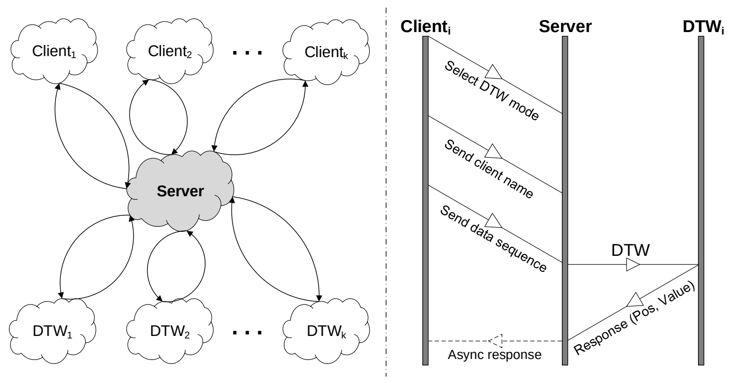

Fig. 1 Left: block diagram of the proposed solution. Right: interchange information protocol among the general modules.

piano performances, however, for instruments with smooth onsets like string and wind it may have difficulties.

Our online DTW approach follows the same strategy as the classic DTW algorithm in order not to add more complexity to the system. However, as it is an online approach the backward tracking of the classic DTW is not computed. The process takes each sequence position with the minimum accumulated cost up to the current frame from only the forward step of the DTW algorithm. Regarding the computational cost of the proposed system, at each frame (whose hop size has been set to 10 ms) the feature extraction and the DTW forward step should be computed. Then, as the score length is file dependent, a fast enough implementation of the DTW step is needed. Also, it should be ready and robust for a wide range of score length files.

3 Software architecture system

As we set in previous sections, the aim of this work is to develop a parallel online DTW software which allows multiple clients, either operating in alignment or accompaniment mode, and running concurrently in different systems with, maybe, different architectures. To do this, our proposal follows a multi-tier architecture, i.e. a *client-server* architecture where its tiers or modules (see Fig. 1) are:

Server: It is responsible for managing the different clients and for running the DTW option associated with each client. Since there may be different clients running concurrently, each client stores its own information in a data structure, which is separated from the information related to other clients.

Client: Each musical instrument has an associated client. At every time slot the client samples for a new sequence (the acquired sound signals or frames after applying preprocessing audio-to-score stage) of the instrument information, sends it to the server via a TCP socket, and waits for the answer. The client's answer is made of an ordered pair *(pos, value)*, where *pos* is the position in the score at the current time. The client-server scheme proposed follows a simple protocol whose basic skeleton is depicted in Fig. 1. In our protocol, the client builds a sequence starting with a header that gathers information about the client identifier and the DTW mode selected. Then, the sequence is dispatched to the server which, in turn, answers back to the client asynchronously.

DTW: This module processes the sequences received from the server using intermediate data that have been updated over time. It is the computational kernel of the software. In other words, client and server tiers handle inputs/outputs, manage data structures, perform communications, etc.

The software has been developed using parallel programing, therefore, current multi-core architectures (x86, x64 and ARM®) and devices (servers, smatphones, tablets, etc.) are fully supported; moreover, they can be used simultaneously during the execution of the software. For the case in which there is only one client, e.g. when using mobile devices or when performing an individual rehearsal, we have built a specific version called *Full_Client*. *Full_Client* is merely the result of combining the above mentioned general modules all together in a single monolithic structure, where we get rid of all communications based on sockets.

The above described architecture has several advantages. For instance, this architecture allows multiple clients to coexists in the same hardware system where the server is running thanks to the fact that the structure of these clients is very light, i.e. the clients are limited to send/receive information. The system resources are under control thus allowing to do score following under very strong "tempo" constraints. Furthermore, this structure allows to have concurrent clients running different operation modes, e.g. clients in alignment mode, clients on accompaniment, etc.

As already stated, the online DTW algorithm has two main stages: build the local distance matrix D, and then fill recursively the warping matrix C (*forward step*). The *forward step*, which is the computational kernel, is depicted in Algorithm 1. When the DTW module receives a new sequence, it is firstly checked if the sequence corresponds to a silence, i.e. a period where there is no music playing (line 4). Silence sequences, which are those where the minimum value is found at the first position, are dismissed. For the other sequences, Algorithm 1 computes the cost of each position of the sequence using matrices C and D applying (1) (lines 8-15). Later, matrix C is updated and the position of v having the minimum value is calculated and returned to the server.

Using the fact that the online DTW version does not apply the *traceback step* we can make some optimizations. On the one hand, the maximum step size at each dimension is less than four due to audio constrictions, thereby, we only need to store matrix D as a vector of dimension $\alpha_n + \alpha_m$ (in real applications eight or fewer positions). On the other, when the ith sequence is

Algorithm 1 Performs online audio-to-score alignment

```
Require: Matrix $D$, $\alpha_n$ and $\alpha_m$ (maximum step size at each dimension) and other global
    parameters and structures
Ensure: The cost of minimum cost path
 1: Setup matrix $C$ and build local structures
 2: repeat
 3:   Receive in vector $v$ the new sequence from the server
 4:   ($min$, $pos$)= find the minimum value of $v$ and its position
 5:   if $pos = 0$ then
 6:     return ($pos$, $min$) to the server
 7:   else
 8:     for $i = 1$ to length($v$) do
 9:       for $j = 1$ to $\alpha_n$ do
10:         Update $v(i)$ using matrices $C$ and $D$
11:       end for
12:       for $j = 1$ to $\alpha_m$ do
13:         Update $v(i)$ using matrices $C$ and $D$
14:       end for
15:     end for
16:     ($min$, $pos$)= find the minimum value of $v$ and its position
17:     Update matrix $C$ using $v$
18:     return ($pos$, $min$) to the server
19:   end if
20: until (number of sequences)
```

processed only the last α_m columns of matrix C are needed, what allows us to use a circular buffer of dimensions $N \times \alpha_m$ to store matrix C. Hence, line 17 of Algorithm 1 performs one column shift over C to the left, and then stores v in the last column of C.

As for the parallelization of Algorithm 1, we point out the following. First, to avoid the *branch divergence* problem we use padding in matrix C. Second, since matrix D is stored into a small vector, the inner loops (lines 9-14) are fused and vectorized. Third, the main loop (line 8) is parallelized using either the OpenMP or the SIMT (Single Instruction, Multiple Thread) model depending on the target architecture. Finally, the reduction operations (lines 4 and 16) have also been parallelized.

From the theoretical point of view, the sequential cost of the computation applied to each input sequence (lines between 4 and 16) can be approximated by

$$T_s(N, N_c) = N + 4NN_c + N = 2N(1 + 2N_c) \approx 4NN_c \,, \tag{3}$$

where N is the number of states per sequence (the size of vector v), and N_c is the number of costs ($\alpha_n + \alpha_m$). We assume that the number of flops for reduction operations is $\approx N$ (lines 4 and 16) when they are performed sequentially, or is $\approx N/p + f(p)$ when they are performed in parallel, being p the number of cores. The actual cost of $f(p)$ depends on the underlying architecture, being $f(p) = \log(p)$ in the best case.

Using the sequential cost in (3) and taking into account our model for a reduction operation in parallel, the expression derived for execution time in parallel has the form

$$\begin{aligned} T_p(N, N_c, p) &= \left(\frac{N}{p} + f(p)\right) + \left(\frac{4NN_c}{p}\right) + \left(\frac{N}{p} + f(p)\right) \\ &= \frac{2pf(p) + 2N(1 + 2N_c)}{p} \approx \frac{2pf(p) + 4NN_c}{p} , \end{aligned} \tag{4}$$

and the efficiency of the parallel version can be expressed as

$$E(M, N, N_c, p) \approx \frac{2NN_c}{pf(p) + 2NN_c} . \tag{5}$$

As already been said, in real applications the term N_c is a small constant so the computational complexity per sequence can be approximated to $O(N)$ for the sequential algorithm and to $O\left(\frac{log(p)+N}{p}\right)$ for the parallel one in the best case. The typical length of N varies between a few hundreds and hundreds of thousands of samples, depending on the composition length. According to the expresion for the efficiency it is clear that when both the composition is only of some few seconds long and the number of cores p is large the efficiency drops sharply. This is the scenario we face, e.g. with hardware accelerators.

4 Evaluation and Experimental Results

We have carried out two different types of experiments. Firstly, we have used the database proposed in [4] (also used in [11]) to validate the proposed system. The database consists of 10 J.S. Bach four-part chorales with the corresponding aligned MIDI data. The audio files are approximately 30 seconds long and are sampled at 44.1 KHz from real performances (see [4] for more information). The second experiment was carried out on a synthetic database to analyse the performance of the application. The durations of the synthetic audio files vary from a few seconds to some hours. The results obtained are shown classified by architecture in the following subsections.

4.1 x86/x64 architecture devices

We used two kind of x86/x64 architecture devices: a) standard CPUs, and b) current coprocessor. As a representative of the former, we use a server with 2 Intel® Xeon® E5-2603 v3 @ 1.60GHz processors with 6 cores each. The Theoretical Peak double precision floating point Performance (TPP) of this machine is about 307 GFlops (26 GFlops per core). The HyperThreading and the Turbo Boost are both deactivated.

For the second kind of the x86/x64 architecture, our testbed is an Intel® Xeon Phi™ 31S1P coprocessor. This device is based on the Intel® Many Integrated Core architecture and has 57 in-order processors with 4 hardware threads each running at 1.1GHz. The TPP is close to 1003 GFlops. All our

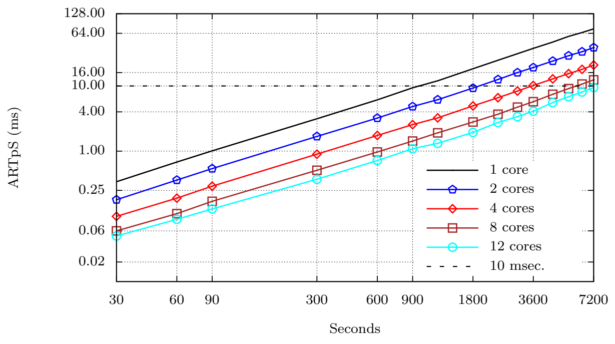

Fig. 2 Evolution of Intel® Xeon® E5-2603 ARTpS.

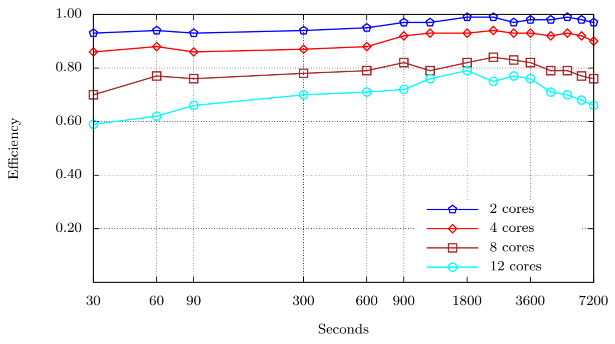

Fig. 3 Efficiency of Intel® Xeon® E5-2603 ARTpS.

experimentation uses the so called "native mode", i.e. the application runs directly on the Intel® Xeon Phi™ coprocessor and its embedded Linux operating system. Thus, there is not data transference through the PCI-e bus, i.e. the incoming samples are directly captured by the coprocessor.

Firstly, we show in Fig. 2 the Intel® Xeon® E5-2603 AveRage Time per Sequence. It is also represented in the figure a flat line for 10 ms corresponding to the maximum acceptable value for the ARTpS to be within the real-time threshold. We can observe that, with 2 cores, the ARTpS is lower than 10 ms for compositions of up to two hours (7200 sec), and how the composition length can be larger as we increase the number of cores used. (Both axes for all plots representing time (ARTpS) are expressed in logarithmic scale for clarity. Also, the x-axis in graphics for efficiency is expressed in this scale.) The results obtained for the efficiency are shown in Fig. 3, where we can observe that these figures are coherent with (5), and that the efficiency is always above 60%.

Respect to the Intel® Xeon Phi™, Fig. 4 (left) shows that the ARTpS is always larger than 10 ms with few cores. The 4.4 GFlops of TPP per core of the Intel® Phi™ is very low compared with the performance of each core (26 GFlops) of the Intel® Xeon E5-2603. Also, we can observe that, irrespective of

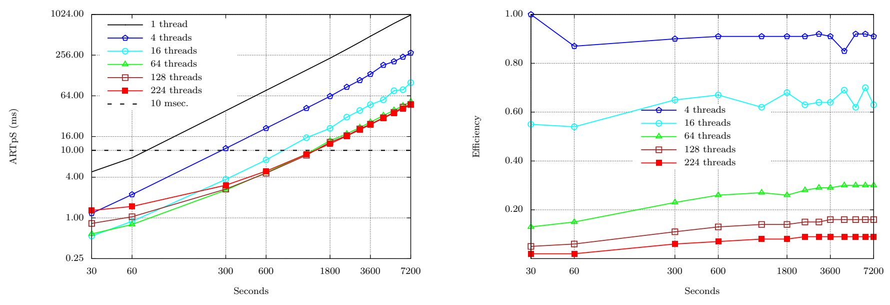

Fig. 4 Evolution of the ARTpS (left) and efficiency (right) on the Intel® Xeon Phi™.

the number of cores, the ARTpS is always greater than 10 ms when processing musical pieces of length longer than 30 minutes (1800 sec). This behaviour is also coherent with the TPP of the target machine since these algorithms involve two operations of reduction per sequence, which implies accessing shared variables and/or communications whose cost, for a fix problem size, increases with the number of cores (see (5)). As it can be observed in Fig. 4 (right), the efficiency is negatively influenced by these reductions. We also observed in our experiments that regardless of the sequence size, the ARTpS grows from 16 cores (4 processors) onwards. Consequently, the suitable use for the Intel® Xeon Phi™ is to execute concurrently 14 DTW online algorithms, keeping one of the 57 processors for control and management tasks. This strategy would result in a better performance and efficiency.

4.2 ARM® architecture devices

This section deals with different devices which all have in common a base processor that implements the ARM® architecture.

The first one is the smartphone S3 of the Chinese company Jiayu. This device contains a processor MediaTek MT6752 @1.7GHz, a 64-bit octacore 4G LTE platform based on the ARM® Cortex®-A53 processor with a GPU ARM Mali™-T760 MP2 700MHz. The mobile operates under Android operating system.

In Fig. 5 (left) is shown the ARTpS when vary both the problem size and the number of cores used. The behaviour is stable and according to the theoretical performance of the device. Also, the efficiency corresponds to the theoretical one (Fig. 5 (right)).

The next device is a NVIDIA Shield tablet. This tablet features the processor Tegra K1 @2.2GHz, which is an implementation of the ARM® Cortex® A15 architecture with 4 cores. This device also operates under Android. The experimental results corresponding to this device are shown in Fig. 6 for the time (left) and efficiency (right).

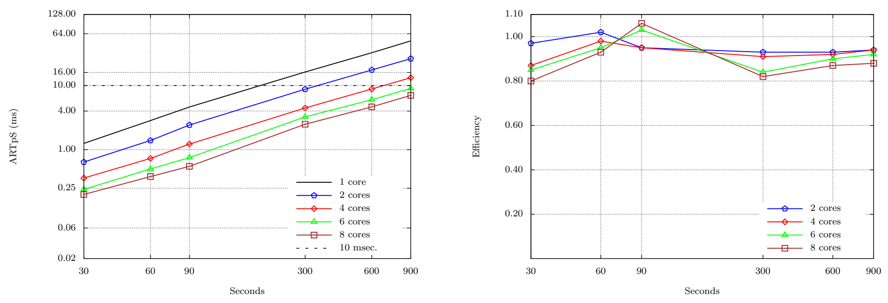

Fig. 5 Evolution of the ARTpS (left) and efficiency (right) on the Jiayu S3.

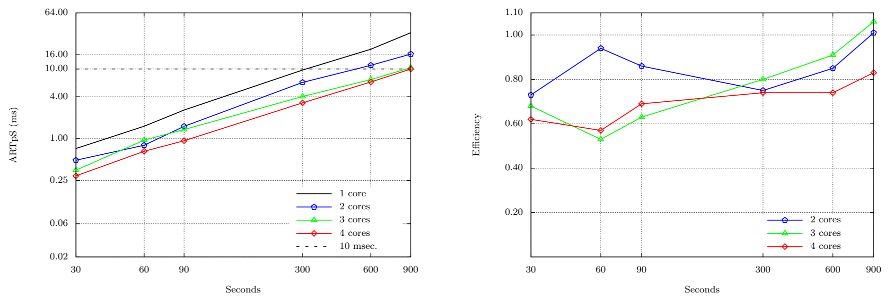

Fig. 6 Evolution of the ARTpS (left) and efficiency (right) on the NVIDIA Shield.

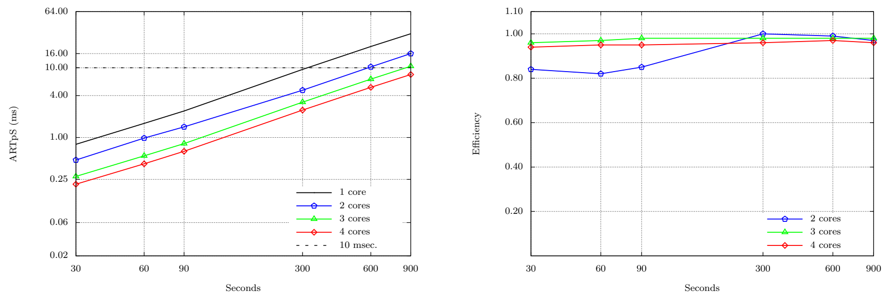

Fig. 7 Evolution of the ARTpS (left) and efficiency (right) on the NVIDIA Jetson Toolkit.

Our last ARM® testbed is a Jetson TK1 development kit. This hardware, powered by NVIDIA, shares everything with the NVIDIA Shield tablet yet, it operates at 2.32GHz and runs a version of Linux operating system (ubuntu 12.04) specially tailored to this device. There also exist other differences regarding the number and type of peripheral connections that do not affect our tests. As in the previous cases we show (Fig. 7) the time (left) and the efficiency (right) of the Jetson.

In view of the results shown in this section it can be concluded that the performance of all ARM-based testbeds used is as expected according to their

TPP and (5). As with the x86/x64 architecture processors, it is possible to obtain an ARTpS lower than 10 ms by using parallel computing in all the ARM-based devices, being this value the cut-off threshold to be considered as a real-time solution.

Finally, the Shield tablet and the Jetson TK1 development kit both use the same TK1 processor containing a power-efficient NVIDIA Kepler$^{\text{TM}}$-based GPU @852 MHz multiprocessor with 192 CUDA cores embedded into the processor die, thus having direct access to data stored into the main memory. This motivated us to implement a CUDA kernel for the GPUs. Our first experiments with this kernel running on Tesla K family devices (attached through a PCIe link to a host) resulted in quite bad numbers, fulfilling thus the theoretical prediction. Yet, contrary to our expectations, the results on the Tegra K1 GPU, on both the Shield tablet and the Jetson, did not outperform the results obtained using only the ARM cores of the same chip, a fact which make us discard the use of GPUs to compute the online DTW.

5 Conclusions and Future Work

There exist two approaches to tackle the *Audio-to-score* alignment problem, often called "offline" and "online". Offline "looks into the future" when establish the matching, reaching higher matching precision, but it is not suitable when real-time audio-to-score alignment is needed as, for example, in automatic page turning or automated computer accompaniment of a live soloist. A well known method and extensively used in the literature for the second stage of *Audio-to-score* alignment is Dynamic Time Warping (DTW). DTW has the advantage of being computationally simple and can be applied to *Audio-to-score* synchronization.

In this work we have presented and analysed a parallel online DTW version. We have chosen the online approach and have incorporated parallel computing techniques because the computation in real-time is critical for our applications. The developed software follows a client-server architecture that allows, e.g. multiple clients to run concurrently in the same system, to run different operation mode (alignment mode or on accompaniment), etc.

The current version of our application has been implemented for multicore architectures, encompassing x86/x64 processors, x64 coprocessors like the Intel® Xeon Phi$^{\text{TM}}$, and ARM-based processors. Thus, the application can be executed in a wide range of mobile devices.

Our proposal is a useful tool for the *Audio-to-score* alignment problem. Furthermore, the framework achieves good score alignments within the real-time constraints of 10 ms by using parallel computing techniques. To support these claims, we have performed a very extensive experimentation that reaches many different devices operating either under Linux or Android.

Acknowledgements This work has been partially supported by Spanish Ministry of Science and Innovation and FEDER under projects TEC2012-38142-C04-01, TEC2012-38142-

C04-03, TEC2012-38142-C04-04, TEC2015-67387-C4-1-R, TEC2015-67387-C4-3-R, TEC2015-67387-C4-4-R, the European Union FEDER (CAPAP-H5 network TIN2014-53522-REDT), and the Generalitat Valenciana under grant PROMETEOII/2014/003.

References

1. C. Joder, S. Essid and G. Richard, “A conditional random field framework for robust and scalable audio-to-score matching,” *IEEE Trans. Speech, Audio and Lang. Process.*, vol. 19, no. 8, pp. 2385–2397, nov. 2011.
2. R. J. McNab, L. A. Smith, I. H. Witten, C. L. Henderson, and S. J. Cunningham, “Towards the digital music library: tune retrieval from acoustic input,” *in DL 96: Proceedings of the first ACM international conference on Digital libraries.* New York, NY, USA: ACM, pp. 11-18, 1996.
3. R. B. Dannenberg, “An intelligent multi-track audio editor,” *in Proceedings of International Computer Music Conference (ICMC)*, vol. 2, pp. 89-94, 2007.
4. Z. Duan, and B. Pardo, “Soundprism: An Online System for Score-informed Source Separation of Music Audio,” *IEEE Journal of Selected Topics in Signal Process.*, vol. 5, no. 6, pp. 1205–1215, 2011.
5. S. Dixon, “Live tracking of musical performances using on-line time warping,” *in Proc. International Conference on Digital Audio Effects (DAFx)*, Madrid, Spain, pp. 92–97, 2005.
6. N. Orio and D. Schwarz, “Alignment of monophonic and polyphonic music to a score,” *in Proc. International Computer Music Conference (ICMC)*, pp. 129–132, 2001.
7. I. Simon, D. Morris, and S. Basu. “MySong: automatic accompaniment generation for vocal melodies”. *In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* ACM, New York, NY, USA, 725-734, 2008.
8. F.J. Rodriguez-Serrano, Z. Duan, P. Vera-Candeas and B. Pardo “Online Score-Informed Source Separation with Adaptive Instrument Models”. *Journal of New Music Research*, Volume 44, Issue 2, London, 2015.
9. A. Arzt, G. Widmer, and S. Dixon. “Automatic Page Turning for Musicians via Real-Time Machine Listening”. *In Proceedings of the 18th European Conference on Artificial Intelligence* IOS Press, Amsterdam, The Netherlands, pp. 241-245, 2008.
10. J.J Carabias-Orti, F.J. Rodriguez-Serrano, P. Vera-Candeas, F.J. Canadas-Quesada, N. Ruiz-Reyes, ”An audio to score alignment framework using spectral factorization and dynamic time warping” 16th International Society for Music Information Retrieval Conference, pp. 742–748, 2015.
11. F.J. Rodríguez-Serrano, J. Menéndez-Canal, A. Vidal, F.J. Cañadas-Quesada, R. Cortina “A DTW based score following method for score-informed sound source separation”, *Proc. of the 12th Sound and Music Computing Conference 2015 (SMC-15)*, Ireland, pp. 491–496, 2015.
12. J.J. Carabias-Ortí, F.J. Rodríguez-Serrano, P. Vera-Candeas, F.J. Cañadas-Quesada, N. Ruíz-Reyes, “Constrained non-negative sparse coding using learnt instrument templates for realtime music transcription,” *Engineering Applications of Artificial Intelligence*, Volume 26, Issue 7, pp. 1671–1680, 2013.
13. C. Raphael, “Aligning music audio with symbolic scores using a hybrid graphical model,” *Machine Learning*, vol. 65, pp. 389-409, 2006.
14. Schreck-Ensemble (2001–2004). ComParser 1.42. `http://home.hku.nl/~pieter.suurmond/SOFT/CMP/doc/cmp.html` (Last visited on september, 2015).
15. F. Itakura, “Minimum prediction residual principle applied to speech recognition,” *IEEE Transactions on Acoustics, Speech and Signal Processing,* vol. 23, pp. 52–72, 1975.
16. R. Dannenberg and N. Hu, “Polyphonic audio matching for score following and intelligent audio editors,” *In Proceedings of the International Computer Music Conference*, San Francisco, International Computer Music Association, pp. 27–34, 2003.
17. Mueller, M., Kurth, F., and Roeder, T. “Towards an efficient algorithm for automatic score-to-audio synchronization,” *In Proceedings of the 5th International Conference on Music Information Retrieval*, Barcelona, Spain. 2004.

18. Mueller, M., Mattes, H., and Kurth, F. “An efficient multiscale approach to audio synchronization,” *In Proceedings of the 7th International Conference on Music Information Retrieval*, Victoria, Canada. 2006.
19. Kaprykowsky, H. and Rodet, X. “Globally optimal short-time dynamic time warping applications to score to audio alignment.” *In IEEE ICASSP*, pp. 249–252. Toulouse. France. 2006.
20. C. Fremerey, M. Müller, and M. Clausen, “Handling repeats and jumps in score-performance synchronization,” *in Proc. ISMIR*, pp. 243–248, 2010.
21. A. Arzt and G. Widmer, “Towards effective any-time music tracking,” *in Proc. Starting AI Researchers Symp. (STAIRS)*, Lisbon, Portugal, pp. 24–36, 2010.