IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON MOBILE MULTIMEDIA: METHODOLOGY AND APPLICATIONS

Received December 30, 2019, accepted February 27, 2020, date of publication March 9, 2020, date of current version March 20, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2979489

An Efficient Implementation of Parallel Parametric HRTF Models for Binaural Sound Synthesis in Mobile Multimedia

JOSE A. BELLOCH[1], (Member, IEEE), GERMAN RAMOS[2], (Member, IEEE), JOSE M. BADIA[3], AND MAXIMO COBOS[4], (Senior Member, IEEE)

[1]Departamento de Tecnología Electrónica, Universidad Carlos III de Madrid, 28911 Leganes, Spain
[2]iTEAM Institute, Universitat Politecnica de Valencia, 46022 Valencia, Spain
[3]Departamento de Ingeniería y Ciencia de Computación, Universitat Jaume I de Castello, 12071 Castellon, Spain
[4]Departamento Informàtica, Universitat de València, 46010 Valencia, Spain

Corresponding author: Jose A. Belloch (jbelloc@ing.uc3m.es)

This work was supported in part by the Spanish Ministry of Economy and Competitiveness under Grant RTI2018-097045-B-C21, Grant RTI2018-097045-B-C22, Grant TIN2017-82972-R, and Grant ESP2015-68245-C4-1-P, and in part by the Universitat Jaume I Project UJI-B2019-36.

ABSTRACT The extended use of mobile multimedia devices in applications like gaming, 3D video and audio reproduction, immersive teleconferencing, or virtual and augmented reality, is demanding efficient algorithms and methodologies. All these applications require real-time spatial audio engines with the capability of dealing with intensive signal processing operations while facing a number of constraints related to computational cost, latency and energy consumption. Most mobile multimedia devices include a Graphics Processing Unit (GPU) that is primarily used to accelerate video processing tasks, providing high computational capabilities due to its inherent parallel architecture. This paper describes a scalable parallel implementation of a real-time binaural audio engine for GPU-equipped mobile devices. The engine is based on a set of head-related transfer functions (HRTFs) modelled with a parametric parallel structure, allowing efficient synthesis and interpolation while reducing the size required for HRTF data storage. Several strategies to optimize the GPU implementation are evaluated over a well-known kind of processor present in a wide range of mobile devices. In this context, we analyze both the energy consumption and real-time capabilities of the system by exploring different GPU and CPU configuration alternatives. Moreover, the implementation has been conducted using the OpenCL framework, guarantying the portability of the code.

INDEX TERMS Binaural synthesis, HRTF modeling, GPU, parallel filters, parametric model, interpolation.

I. INTRODUCTION

Applications using spatial audio rendering are gaining popularity, mainly due to the widespread use of multimedia-capable mobile devices such as phones and tablets. These applications include gaming, immersive videoconferencing systems, augmented and virtual reality, and interactive 3D and 360° video and audio playback [1]. The emergence of systems-on-chip (SoC) that contain a small graphics accelerator (or GPU), provides a notable increment of the computational capacity of mobile multimedia devices, while partially retaining the appealing low-power consumption of embedded systems. This is the case, for example, of the Samsung Exynos 9820 SoC [1] that includes a Mali-G76 MP12 GPU and is present, among others, on the Samsung Galaxy Note 10. In fact, multiple current mobile devices contain in their SoC a Mali-based GPU.[2] Most of these devices can be programmed using the popular OpenCL framework [2], [3], which offers the possibility of implementing parallel codes portable to a wide range of platforms, from low-cost mobile devices to high-performance GPUs, modern CPUs or FPGAs. The use of the GPU for applications beyond image-processing allows taking profit of the GPU inherent

The associate editor coordinating the review of this manuscript and approving it for publication was Danda Rawat.

[1]https://en.wikichip.org/wiki/samsung/exynos/9820
[2]https://deviceatlas.com/blog/most-used-smartphone-gpu

parallel architecture in compute-intensive applications so that computational resources of the CPU can be used for other tasks. In fact, there exist multiple audio applications that use the GPU for carrying out their digital signal processing, such as [4]–[6].

Head-related transfer functions (HRTFs) are often used for binaural sound synthesis in those applications that require a spatial audio rendering engine. HRTFs collect all the modifications that a free-field sound wave suffers from its original source to the receiver's tympanic membrane. These modifications on the wave include, among other effects, the inherent acoustic path from source to listener, and the reflection and diffraction patterns form the receiver's own anatomy (shoulders, head size and form, hair, pinna shape). The brain compares the information received at both ears to locate the sound source position [7]. In this context, the interaural level differences (ILD) and the interaural time differences (ITD) are relevant for the localization accuracy on the horizontal plane. Other aspects like the frequency shading created by the diffraction effect of the head (mainly at high frequencies), and the reflection patterns in the pinna and torso depending on the direction of arrival of the sound, play also an important role in the horizontal and vertical planes. All of these details are captured within the HRTFs for each ear and each sound source direction.

There are different methods for obtaining the individual HRTFs sets. It is possible to measure them directly with a specific and complex setup and post-processing algorithms [8]–[10]. Alternatively, they can also be synthetically created by using analytical models or numerical simulations of the complete head and torso [11]. Recently, it has been proposed to construct the HRTFs from several pictures of the head and pinna [12]. Actually, there are several HRTFs databases publicly available for research purposes like the CIPIC [13], LISTEN [14], ARI [15] or ITA [16].

The number of multimedia applications running over mobile devices that are demanding binaural audio is constantly increasing. However, there are some important considerations to be taken into account in a mobile multimedia context, where accurate and realistic low-order HRTF models are desirable for several reasons. First, a low order model requires a lower computational cost. Second, it will drain less battery energy, a crucial aspect of portable devices. Finally, the efficient implementation of low-order models on parallel architectures like GPUs can also have a considerable impact on the final computational cost and energy consumption.

Several approaches have been proposed in the literature to build low-order HRTF models [11], [17], [18]. Some methods employ an analytical model of the head and torso, modeling the propagation delay and the diffraction effects of the wave. Other approaches are based on the design of a digital filter that models the behavior of the original HRTF, such as the one implemented in this paper that was previously developed by the authors [19], [20], based on infinite impulse response (IIR) filters. As will be seen, the HRTF model implemented in this paper has several advantages that makes it suitable to be executed in multimedia mobile devices. On the one hand, it is constructed as a parallel bank of second-order sections (SOS) that could be executed efficiently on the parallel architectures included in mobile devices. On the other hand, it is a parametric model. That means that, instead of defining and storing the filter coefficients of each SOS, physical parameters (frequency, gains, and quality factor Q) are used. This parametric approach, as it will be seen, allows a simple interpolation method for obtaining the HRTFs at azimuth and elevation angles that have not been modeled, and it will need a lower database size for storing the complete HRTF set.

In this paper we describe a new portable OpenCL implementation of the parallel parametric HTRF model. In our development, we combine two levels of parallelism. On the one hand, multiple sound sources are processed in parallel. On the other hand, the processing of each SOS in a filter is also carried out in parallel. We analyze the performance of the algorithm on a low power SoC showing that we can leverage its small GPU to process up to 32 sound sources in real-time while freeing the CPU to run other applications. The scalability shown by the algorithm can greatly increase this number on the more powerful GPUs included on most modern mobile devices. Energy consumption and battery life is a fundamental factor to take into account when designing applications for this kind of devices. Therefore, we evaluated the possibility of regulating the frequency of the CPU and GPU cores in order to reduce the energy consumption of our algorithm. Results show that by using the low power-consuming kind of CPU core (Cortex-A7) as host and lowering the frequency of the CPU and GPU cores, we can greatly reduce the energy consumption, while still being able to process up to 16 sound sources in real-time.

The rest of the paper is organized as follows. Section 2 describes the parametric HRTF model and its evolution from an original SOS chain to a parallel SOS bank. In section 3, several optimizations of the parallel model are presented. These optimizations are carried out for reducing the total computational cost and energy consumption, and for taking the maximum profit of the parallel architecture of the GPU. The details of the experimental environment are described in section 4, while the OpenCL software architecture is detailed in section 5. All the experimental results in performance and energy consumption are presented in section 6. Finally, the conclusions are summarized in section 7.

II. THE PARAMETRIC PARALLEL HRTF MODEL

The parametric HRTF model considered throughout this paper was developed by the authors in a two-step approach. The model, implemented as a SOS chain, was first proposed in [19], demonstrating the benefits of the parametric approach in terms of data storage needs and interpolation simplicity. Later, a parallel adaptation was developed in [20] to overcome the serial-to-parallel conversion while maintaining the desirable properties of the original model. For the sake of a self-contained presentation, the next subsections briefly

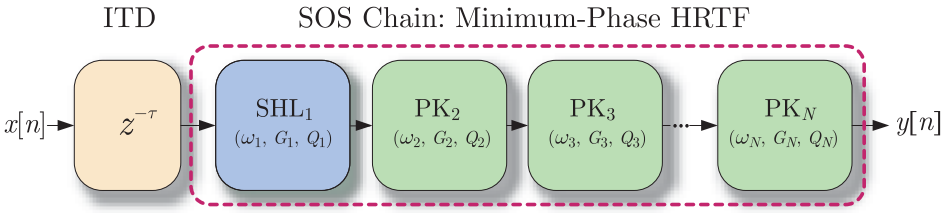

FIGURE 1. HRTF parametric model as a SOS chain.

describe both models, with emphasis on the parallel one as it is the final model optimized, implemented and evaluated in this paper.

A. PARAMETRIC SECOND-ORDER SECTION CHAIN MODEL

This section describes the basic parametric model based on a low-order IIR filter implemented as a chain of SOS [19]. The idea resides in approximating the frequency response of the target HRTF with typical second-order audio digital filters [21], [22], i.e. peak filters and low-frequency shelving filters. The complete model is shown in Fig. 1.

The model consists of a first block that is a delay line whose objective is to implement the ITD part of the HRTF. The value of the number of delay samples, τ, is obtained from the lag corresponding to the maximum of the cross-correlation between the original HRIRs (*Head-Related Impulse Responses*, i.e. the time-domain equivalent of the HRTF) of both ears at the target angles (azimuth ϕ and elevation θ). For a more accurate localization, non-integer values of τ could be implemented using, for example, one of the efficient techniques described at [23] or [24]. After the delay line, there is a N SOS chain that models the minimum-phase frequency response of the HRTF. The first one SHL_1 is a second-order low-frequency shelving filter used to model the low-frequency behavior of the HRTF. Due to the diffraction effect, frequencies below 400 Hz vary mainly only in level, and a low-frequency shelving can mimic this behavior. The rest of the SOS elements in the chain, PK_i ($i = 2, \ldots, N$), are conventional peak audio filters that model the peaks and valleys present in the HRTF frequency response.

Each of the SOS elements is defined by the three parameters: digital center frequency ω_i, gain G_i, and quality factor Q_i. Thus, the complete HRTF at a specific direction and ear is modeled with N sets of parameters (ω_i, G_i, Q_i), ($i = 1, \ldots, N$), and the value of τ, all of them with a direct physical meaning. Each SOS has a transfer function $H_i(z)$ in the z-domain defined as

$$H_i(z) = \frac{b_0^{(i)} + b_1^{(i)} z^{-1} + b_2^{(i)} z^{-2}}{a_0^{(i)} + a_1^{(i)} z^{-1} + a_2^{(i)} z^{-2}}. \quad (1)$$

The relation between the parameters (ω_i, G_i, Q_i) and the filter coefficients ($b_k^{(i)}$ and $a_k^{(i)}$) can be found at [19], [21], [22]. The parametric approach arises from the use of the set of parameters instead of the five filter coefficients of each SOS.

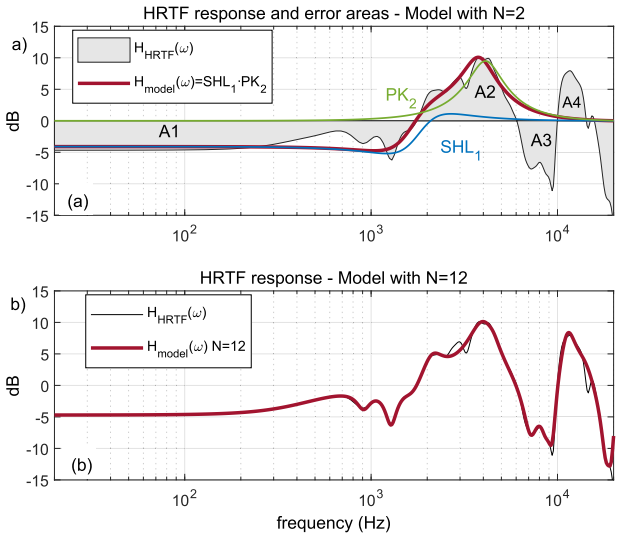

FIGURE 2. Original HRTF(ω) and approximated response $\text{HRTF}_{model}(\omega)$.

An off-line iterative method for obtaining the values of the parameters of each SOS is detailed in [19], which is based on a perceptually motivated cost function. Such cost function is the absolute decibel error with respect to the original HRTF smoothed $1/12^{\text{th}}$ and evaluated over a discrete logarithmic frequency axis, with a resolution of $1/48^{\text{th}}$ octave. Fig. 2(a) shows the iterative process for $N = 2$ where A1 to A4 are the error areas (differences between the target HRTF and the actual modeling filter state). For each SOS, first, a good set of initial values of the parameters ($\omega_{i'}$, $G_{i'}$, $Q_{i'}$) is specified trying to fit the biggest error area. Then, these initial values are later optimized iteratively. SHL_1 (in blue) tries to cancel A1. Then, once applied SHL_1, the peak filter PK_2 (in green) continues with the next bigger error area A2, and so on. The achieved model with $N = 2$ is displayed in red. Once all the SOS have been designed, a final post-optimization stage is carried out to improve the interaction among SOS with neighbouring frequencies. The obtained HRTF model with $N = 12$ is shown in red in Fig. 2(b), approximating the original HRTF. Such a model would need only 37 parameters, 12 sets of ($\omega_{i'}$, $G_{i'}$, $Q_{i'}$), and the τ value, that have physical meaning, instead of the 200 coefficients of the impulse response of the original HRTF. More details can be found at [19].

To create the complete set of HRTFs, once the model is obtained at $\phi = 0°$ and $\theta = 0°$, the parameters for the next

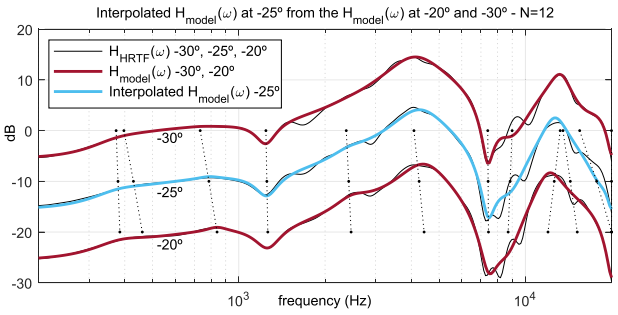

FIGURE 3. Interpolated $HRTF_{model}(\omega)$ at $-25°$ form the models at $-20°$ and $-30°$ with $N = 12$.

directions in azimuth and elevation are obtained by modifying iteratively the already calculated parameters as a starting point. This procedure works well because the frequencies of the peaks and valleys of neighbouring positions do not change abruptly.

One of the major benefits of this parametric approach is the simplified interpolation procedure that can be used for synthesizing HRTFs at directions that have not been modeled before. This is a common problem, where different solutions have been proposed, always with a significant increase in computational cost [25], [26]. With the parametric approach, this can be easily achieved by interpolating the values of the model's parameters. As an example, Fig.3 shows the interpolated response at $\phi = -25°$ and $\theta = 0°$, obtained from the modeled responses at $\phi = -20°$ and $\phi = -30°$, both with $\theta = 0°$ and $N = 12$. Responses are scaled at the figure 10dB for clarity. The vertical dashed lines display the values of the frequencies of the SOS at the modeled HRTF. Frequencies of each SOS at $\phi = -25°$ are obtained by interpolation. The same for the gains, Qs, and the value of τ.

This parametric approach has proven with subjective tests [19] that with $N = 12$ (i.ei. 37 parameters per direction), it is possible to maintain the perceptual characteristics of the original HRTFs (with 200 stored coefficients). Even with $N = 6$ (19 parameters per direction), satisfactory results are reached for most applications. At the same time, as the interpolation procedure is efficient and simple, it is even possible to decrease the number of directions to be modeled at least to a half. As a consequence, an order-of-magnitude reduction in the total database size can be obtained, guarantying the perceptual quality.

B. PARALLEL MODEL BY PARTIAL FRACTION EXPANSION

The previous parametric HRTF model implemented with a second-order chain performs well considering both the computational cost involved and the perceptual quality. However, the data dependencies arising from its serial structure does not allow to implement it efficiently on parallel architectures such as GPUs. The simplest way to overcome this problem is transforming the already designed N SOS chain into a mathematically equivalent parallel filter bank using the well-known partial-fraction-expansion (PFE) [27]. To accomplish that, the coefficients $b_k^{(i)}$ and $a_k^{(i)}$, $k = 0, 1, 2$, can be extracted from the parameters of each SOS and normalized by $a_0^{(i)}$. Then, the parallel conversion is carried out by transforming the original transfer function $H(z)$, a product of SOS, into an addition of a constant K and biquadratic sections

$$H(z) = \prod_{i=1}^{N} \frac{b_0^{(i)} + b_1^{(i)} z^{-1} + b_2^{(i)} z^{-2}}{1 + a_1^{(i)} z^{-1} + a_2^{(i)} z^{-2}} = K + \sum_{i=1}^{N} \frac{b_0'^{(i)} + b_1'^{(i)} z^{-1}}{1 + a_1'^{(i)} z^{-1} + a_2'^{(i)} z^{-2}}. \quad (2)$$

Resolving PFE, the new coefficients K, $b_k'^{(i)}$, $(k = 0, 1)$, and $a_k'^{(i)}$, $(k = 0, 1, 2)$ are calculated, obtaining a mathematically equivalent transfer function expressed in a parallel form. The data dependence between the consecutive stages in the SOS chain is not present in the resulting parallel form. By following such an approach, it is possible now to calculate in parallel all the sections on a parallel processor. The structure of this parallel model is shown in Fig. 4.

The sequential implementation of Fig. 1, and its direct conversion to parallel by PFE of Fig. 4, are, by definition, mathematically identical. Also, their computational cost is similar, being slightly lower in the parallel model due to the fact that it only needs 4 multiplications per biquadratic section instead of five in the sequential one. However, one of the major benefits of the parametric approach is lost in this serial to parallel conversion, as the responses can not be easily interpolated anymore with the new structure.

To illustrate this problem, an example extracted from [20] is shown in Fig. 5(a), where the red line indicates the result of the parallel model by PFE obtained from the sequential model with $N = 6$. The original response corresponds to the left HRTF of subject 003 in the CIPIC database [13] for $\phi = 5°$ and $\theta = 0°$. The value of K from Eq. 2 is displayed in green, while the contributions of each biquadratic section are in blue. Changing the modeled position from $\phi = 5°$ to $\phi = 10°$ gives the frequency responses of 5(b). Although the responses are quite similar due to the slight change in azimuth, significant changes of up to 20 dBs are obtained for the K values and the gains of the individual biquadratic sections. In contrast to the sequential implementation, now the phase relations between the parallel biquads are relevant. These large differences do not allow us to synthesize new responses by using a direct interpolation of the parameters, as in the case of the sequential parametric model.

C. PARAMETRIC PARALLEL MODEL

As previously discussed, while PFE provides a parallel equivalent of the sequential model that can be efficiently implemented on a GPU, synthesizing new responses from modeled directions by direct interpolation of the parameters is not possible. To solve this issue, the authors proposed in [20] a parallel version for HRTF modeling following the same parametric philosophy of the serial implementation. Parallel filter bank implementations in audio applications are common [28]–[31], being one of their benefits their better

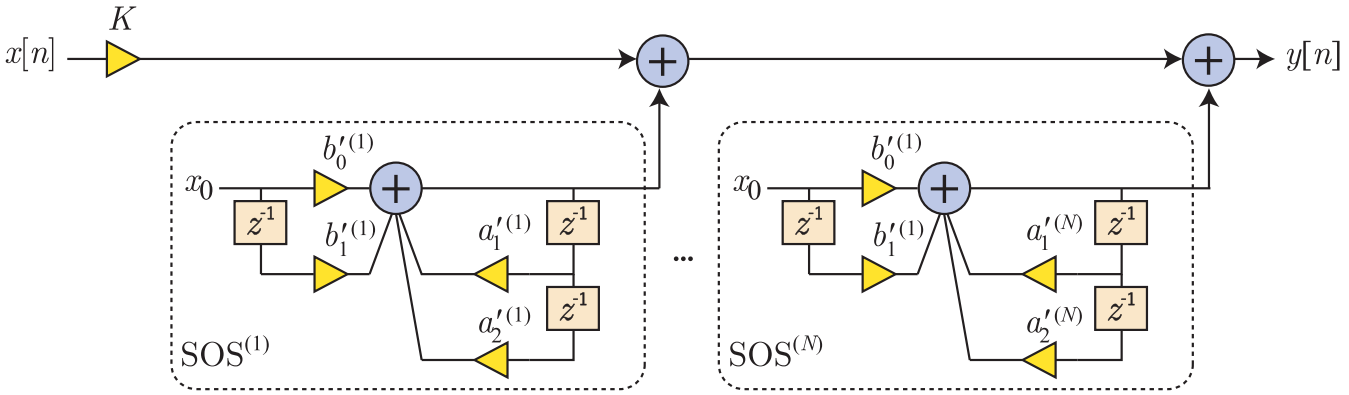

FIGURE 4. Parallel conversion by PFE.

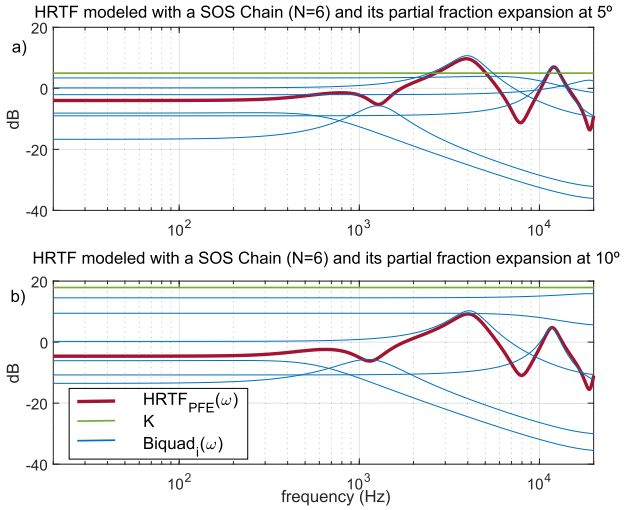

FIGURE 5. Partial-fraction-expansion with $N = 6$ SOS at two consecutive positions.(a) $\phi = 5°$. (b) $\phi = 10°$.

signal-to-noise ratio properties [32]. This parametric parallel implementation adds the advantage of allowing for a simple interpolation while reducing, even more, the computational cost and storage needs of the binaural rendering engine.

The structure of the parametric parallel filter is displayed at Fig. 6. Compared to the sequential structure of Fig. 1, the delay line for the ITD part remains identical, but the serial SOS chain, now it is a parallel SOS bank. LP_1 is a second-order low-pass filter that, in parallel connection with the direct signal path, replaces the low-frequency shelving filter SHL_1 of the serial implementation. The same happens with the new band-pass filters BP_i ($i = 2, \ldots, N$). They replace the previous peak filters PK_i ($i = 2, \ldots, N$). A peak filter can be described as a band-pass filter added to a direct signal path. This parallel model of Fig. 6 follows the parametric approach of the original sequential model, as all the SOS depend again on the sets of parameters (ω_i, G_i, Q_i). The ITD value τ remains the same as in the original sequential implementation. As a result, the simple interpolation feature is preserved, but now with a parallel filter bank structure that can be readily implemented on a GPU architecture. As will be seen, another benefit of this parallel implementation is that the band-pass SOS filters need two multiplications and accumulations (MACs) less than the peak SOS filters.

Fig. 7 shows in red the achieved HRTF parallel model with $N = 12$, for the same HRTF response of Fig. 2 at $\phi = 0°$. As observed, the quality of the approximation is similar to the one of the sequential model, with deviations of only ± 1 dB, but the new model offers an efficient parallel implementation. The individual SOS responses are also displayed, with a solid line when having positive gains (for creating peaks on the frequency response), or with a dashed line (if the gain is negative, creating valleys in the frequency response).

Compared to the parametric sequential HRTF model with $N = 12$, this parallel implementation needs to increase the number of SOS to $N = 16$ in order to be able to model the HRTF response properly at extreme angles. This is because of the relevance of the phase relation between the parallel SOS, which has no impact on the serial model. Nonetheless, both models have a similar computational cost [20].

To analyze the behavior of the interpolation of the responses in this parallel form, Fig. 8 displays in red the parallel HRTF models obtained with $N = 16$ for $\theta = 10°$, and ϕ from $-30°$ to $30°$ in steps of $10°$. The model at $0°$ was created from scratch, evolving the rest of the responses from it. The responses in blue are at multiples of $5°$ and all of them have been created by interpolation from their neighbouring modeled responses. Note that the tracking and similitude respect to the original HRTFs is satisfactory, with small narrow and local deviations. The dots of the vertical lines represent the values of the frequencies of each SOS. As before, neighboring responses are displayed with a 10 dB offset for clarity.

III. OPTIMIZATION OF THE PARAMETRIC PARALLEL MODEL FOR PARALLEL ARCHITECTURES

This section describes further optimizations developed over the presented parametric parallel model with the aim of maximizing the computational efficiency on parallel architectures, such as the GPUs found in current mobile devices. The idea

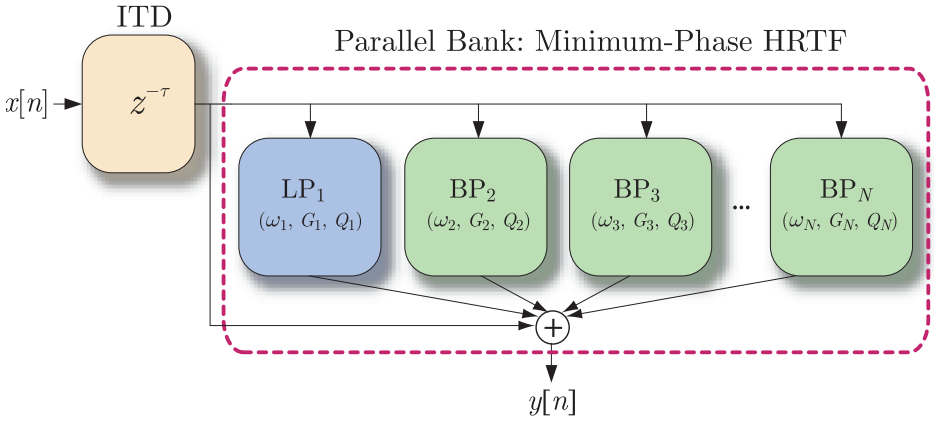

FIGURE 6. Proposed parametric parallel implementation.

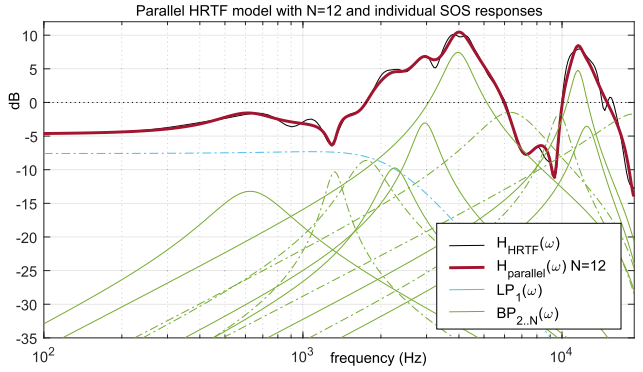

FIGURE 7. Parametric HRTF parallel model with $N = 12$ and the individual SOS responses. Solid lines are SOS with positive gains. Dashed lines represent SOS responses with negative gains.

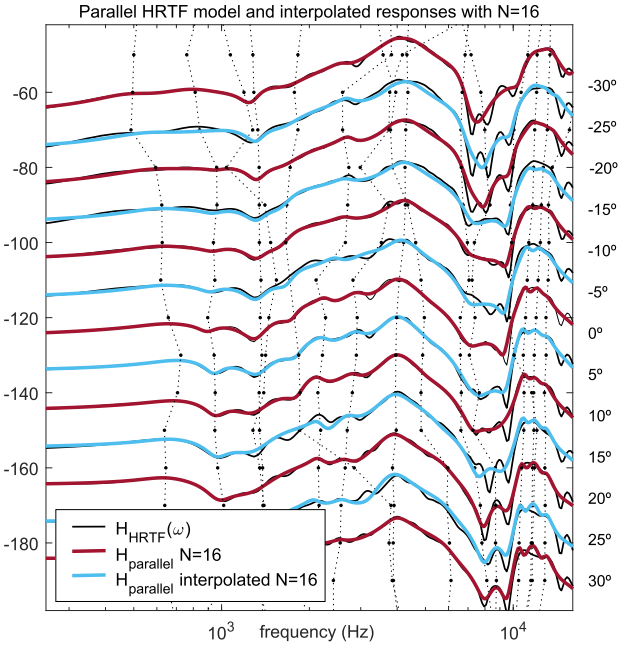

FIGURE 8. Modeled and interpolated HRTFs evolved from $\phi = 0°$. Lines with dots represent SOS frequencies.

is to execute in parallel (using OpenCL work-items) the same algorithmic structure, in this case, each SOS of the proposed model in Fig. 6. Thus, the objective is to reduce the number of operations to be performed in each SOS.

Grouping the coefficients of Eq. (1) in vectors $\mathbf{b}^{(i)} = [b_0^{(i)}, b_1^{(i)}, b_2^{(i)}]^T$ and $\mathbf{a}^{(i)} = [a_0^{(i)}, a_1^{(i)}, a_2^{(i)}]^T$, for the band-pass SOS BP_i, their values could be calculated with the next design formulas that are a function of their parameters (ω_i, G_i, Q_i) [21]:

$$\mathbf{b}_{LP}^{(i)} = G_i \cdot \begin{bmatrix} \alpha_i \\ 0 \\ -\alpha_i \end{bmatrix} \tag{3}$$

$$\mathbf{a}_{BP}^{(i)} = \begin{bmatrix} 1+\alpha_i \\ -2\cdot\cos(\omega_i) \\ 1-\alpha_i \end{bmatrix} \tag{4}$$

where $\alpha_i = \sin(\omega_i)/(2Q_i)$. Assuming from now on the classical normalization of the coefficients by the term $a_0^{(i)}$, the vector in (3) can be expressed as

$$\mathbf{b}_{LP}^{(i)} = b_0^{'(i)} \cdot \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} \tag{5}$$

with $b_0^{'(i)} = \alpha_i \cdot G_i/(1+\alpha_i)$. Now there is a scale factor $b_0^{'(i)}$ that multiplies the term $(1 - z^{-2})$ in the numerator. On the one hand, there is no b_1 coefficient, so one MAC is saved. On the other hand, as $b_2^{(i)} = -b_0^{(i)}$, the scaled numerator term, $(1 - z^{-2})$, can be moved out after the addition of the outputs of all of the band-pass SOS. This new structure is shown in Fig. 9. All the BP_i $(i = 2, \ldots, N)$ have their inputs scaled by $b_0^{'(i)}$ before doing their accumulations. Then, the common $(1 - z^{-2})$ is performed after this accumulation only once. As a result, each band-pass SOS only requires three MACs.

Regarding the low-pass SOS of Fig. 6 LP_1, its $\mathbf{a}_{LP}^{(i)}$ are identical to the band-pass ones $\mathbf{a}_{BP}^{(i)}$. The vector with the numerator coefficients is now

$$\mathbf{b}_{LP}^{(i)} = G_i \cdot \begin{bmatrix} (1-\cos(\omega_i))/2 \\ 1-\cos(\omega_i) \\ (1-\cos(\omega_i))/2 \end{bmatrix}. \tag{6}$$

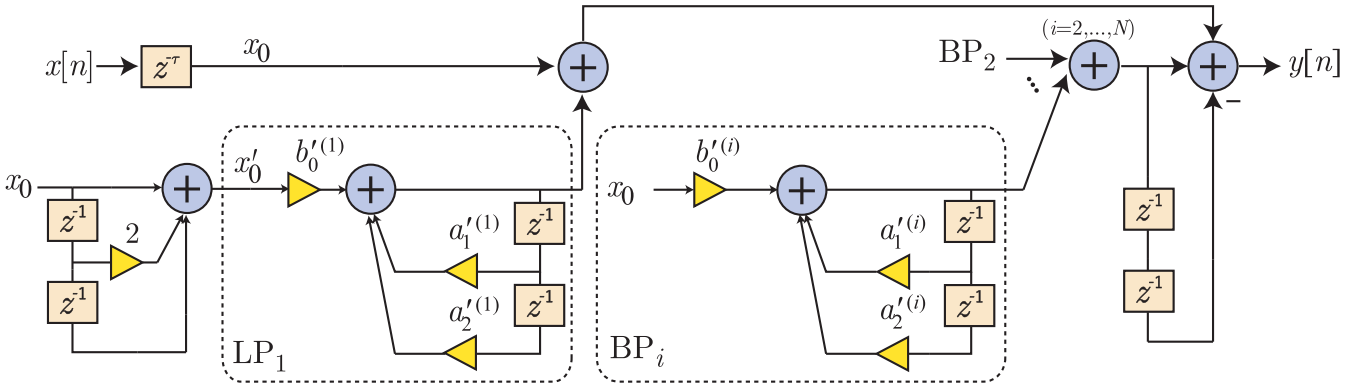

FIGURE 9. Parametric parallel HRTF optimized implementation. LP_1 is a low-pass filter, and BP_2 to BP_N are band-pass filters.

In order to achieve an identical SOS filter structure as the BP_i, after normalization by $a_0^{(1)}$ equation (6) can be organized as

$$\mathbf{b}_{LP}^{(i)} = b_0^{'(1)} \cdot \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}, \tag{7}$$

$$b_0^{'(1)} = G_i \cdot (1 - \cos(\omega_i))/2)/(1 + \alpha_i). \tag{8}$$

With this computational scheme of LP_1, it is possible to have the same structure as the band-pass ones with the three MACs, allowing us to execute the calculus of all the SOS sections concurrently. LP_1 needs to be fed with the numerator stage as seen in Fig. 9. The output of the LP_1 must be added at the final accumulator, not passing through the $(1 - z^{-2})$ that is for the band-pass sections only. Needless to say, that all the calculus of the coefficients $b_k^{(i)}$ and $a_k^{(i)}$ $(k = 0, 1, 2)$ that are a function of the SOS parameter's values (ω_i, G_i, Q_i) will be executed in the CPU of the system and sent to the parallel processing unit (like a GPU). They will vary slowly, depending on the direction to be synthesized. These variations occur at a much lower speed compared with the sampling frequency of the audio signal, so the computational cost demanded from the CPU will be negligible.

In comparison with the sequential implementation [19], this optimized parallel structure provides a computational cost-saving close to 40% and can be efficiently implemented on parallel architectures. Each SOS needs only three MACs instead of five, adding only the one MAC and two additions to pre-filter the LP_1, and the final $(1 - z^{-2})$ part for all the BP_i.

IV. EXPERIMENTAL ENVIRONMENT: GPU AND openCL

Besides the advantages provided by the proposed filtering scheme, additional optimizations can be achieved using parallel processors such as GPUs. That is, not only the filtering process is parallelized by computing independently the SOS of a filter, but we can also filter multiple sound sources simultaneously.

We use the Mali-T628 MP6 GPU included on an ODROID XU3[3] board to test the proposed parallel technique. This GPU is also included in the Exynos 5422 SoC present in the Samsung Galaxy S5 (SM-G900H). This is an older and less powerful version of the Exynos 9820 SoC included in the latest Samsung Galaxy S10, but the results can be approximately extrapolated to the most modern Exynos SoC, since they share the same architecture. Besides the GPU, this specific SoC uses a big.LITTLE heterogeneous computing architecture that combines a quad-core ARM Cortex-A15 and a slower and battery-saving quad-core ARM Cortex-A7. One of the most interesting features of this platform is the possibility of adjusting its energy consumption by reducing the frequency of the CPU cores or the GPU, or even by disabling some of these components. In our experiments, we measured the power dissipation of the Exynos 5422 using the `pmlib` framework [33] to collect the instantaneous power readings from the internal energy-monitoring sensors [34]. To this end, we leverage the four real-time current sensors that can be sampled to obtain the power consumption of four separate power domains of the ODROID XU3 board: Cortex-A15 cores, Cortex-A7 cores, DRAM and Mali GPU.

In order to implement the proposed HRTF filter on the Mali GPU, we use OpenCL (Open Computing Language) framework [2], [3], which enables general-purpose parallel programming across CPUs, GPUs and other kinds of processors. The OpenCL platform model describes the processors as devices composed of different compute units (CU), each of them containing multiple processing elements (PE). In the OpenCL programming model, data-parallelism is achieved by dividing the computation into multiple work-items that can run the same code in parallel on different processing elements over different data. Work-items are combined on work-groups that are executed on different CUs. The code executed by each work-item is included in kernels that are submitted by the host to the devices. Regarding the OpenCL memory model, the memory is divided into four distinct regions: global to each device, constant, local to each work-group and private to each work-item.

Depending on the CPU or GPU device, the distinct memory regions of the model may be mapped to different hardware memories. The use of faster memory in the kernels may drastically improve its performance. We should also take into

[3]https://www.hardkernel.com/shop/odroid-xu3

account the cost of transferring information from the host CPU to the GPU device and the cost of copying information among different memory regions. The Mali-T628 GPU includes two OpenCL devices, `Dev0` with four CUs associated with four of its cores and the `Dev1` consisting of two CUs associated with the remaining two cores.

V. OpenCL IMPLEMENTATION

We have implemented two OpenCL kernels to carry out the filtering process. Firstly, we launch `kFilter`, so that M work-groups process in parallel M sound sources. Next, we use `kAcum` to accumulate the processed samples in the two output buffer of samples that correspond to the left and right channels, respectively. The filter to be executed is composed of a maximum of $N = 16$ SOS, as shown in Figure 9. The sections $H_i(z)$ of the model includes the low-pass LP_1, and the band-pass BP_2 to BP_N.

To render M sound sources using `kFilter`, a total of $2M$ IIR filters must be computed concurrently. Thus, M *work-groups* are launched to run the kernel, one per source. Each *work-group* contains 32 *work-items* and each of them computes a SOS section $H_i(z)$ of the model.

The first 16 *work-items* of the *work-group* are devoted to the computation of the 16 sections corresponding to the left-ear channel, whereas the second 16 *work-items* compute the 16 sections that correspond to the right-ear channel. We are using two-dimensional *work-groups*, each including 2×16 *work-items*. Thus, each *work-group* computes two complete HRTF filters (left and right) associated with one source. Every *work-item* computes one section and stores its result in the local memory shared by all the *work-items* of its *work-group*. Then, a synchronization barrier is set in order to wait until all the *work-items* of the *work-group* have finished. Afterward, a reduction is carried out, where one *work-item* per channel (two in total, one for the left-channel and other for the right-channel) sums up all the values of a vector associated to its channel that is stored in the local memory. Only the pre-FIR filter (shown as x'_0 in Fig. 9) for the low-pass SOS, and the common $(1 - z^{-2})$ for the band-pass SOSs need to be executed before and after the parallel computation of the SOS, respectively. Figure 10 depicts the described operations for one input sound source. It is important to highlight that besides the intrinsic parallelism found within each filter, we are launching concurrently M *work-groups* that can be executed in parallel on different CUs for M different sound sources.

Once we have computed the two output samples per sound source, we need to add them up in two final output samples that will afterward render at the left-channel and the right-channel, respectively. This computation is carried out by the kernel `kAcum` launching in parallel one work-item per sample.

VI. EXPERIMENTAL EVALUATION

This section discusses the experiments conducted to evaluate the execution time and energy consumption of the proposed

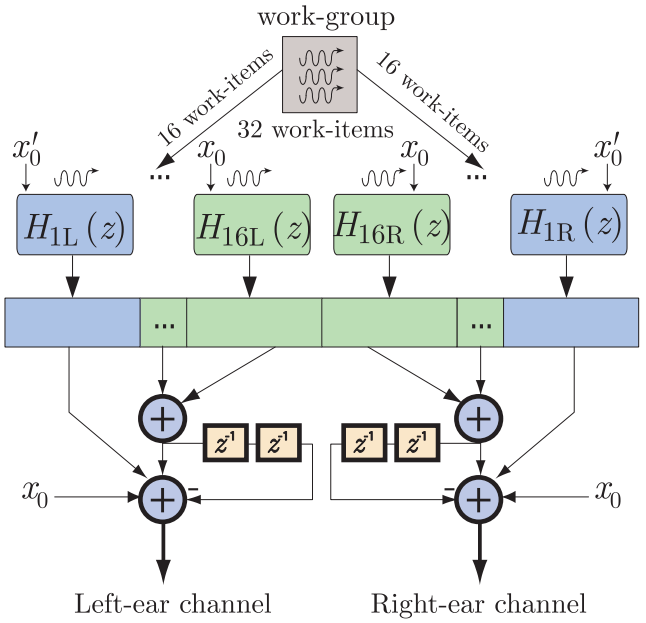

FIGURE 10. Parallel execution carried out by a ***work-item*** inside a ***work-group*** in the kernel `kFilter`.

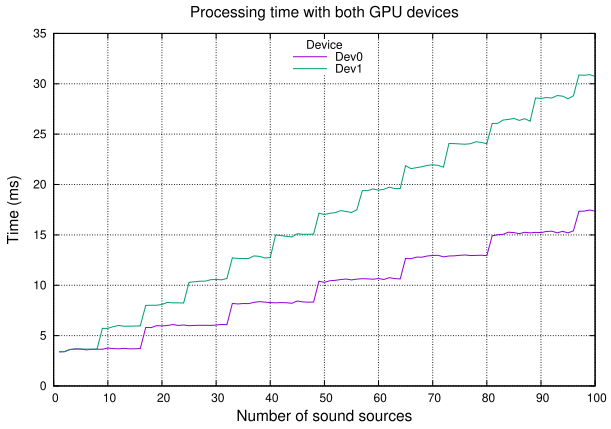

FIGURE 11. Execution time of the algorithm to process 256 samples using both GPU devices.

OpenCL implementation. It is assumed that the audio device provides audio data in buffers of 256 samples per sound source, with a sampling frequency of $f_s = 44.1$ kHz, which imposes a limit of 5.8 ms to process the sources in real-time. It must be taken into account that we deal with $M \times 256$ input samples to compute 2×256 output samples.

The processing time depends on the GPU device as one includes twice the cores of the other. As observed in Fig. 11, the smaller device takes twice the time to process the same number of cores. Therefore, if the more modern Mali G76 is used, which includes 12 improved cores, we would be able to process the same number of sources at least three times faster than with the faster device included on the Mali-T628. We can also see in Fig. 11 the stepped behaviour of the execution time, where the time grows very slowly during a fixed range of sources and then increases abruptly to the

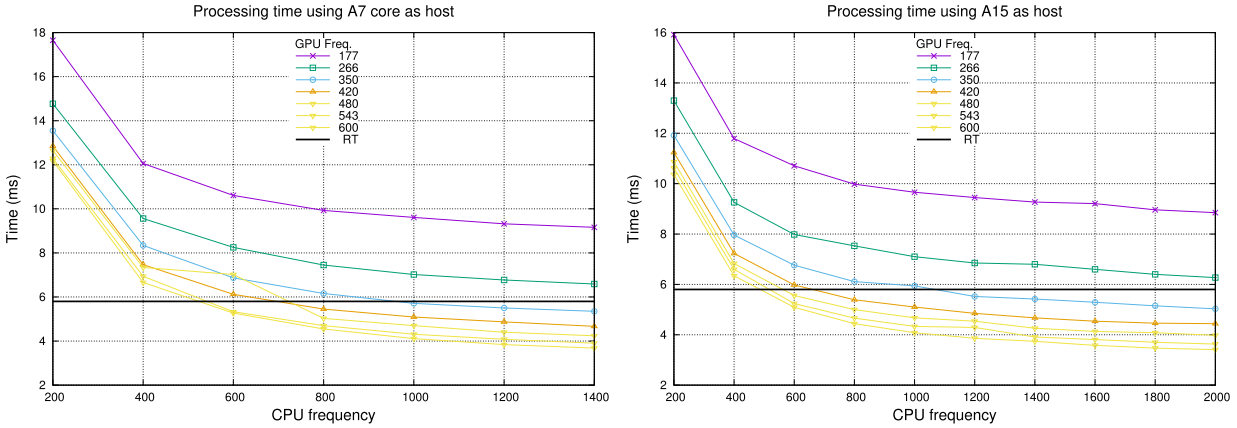

FIGURE 12. Execution time of the algorithm to process 256 samples from 8 sources using both kinds of CPU cores as host of the OpenCL algorithm. The horizontal line represents the time to process the sources in real-time.

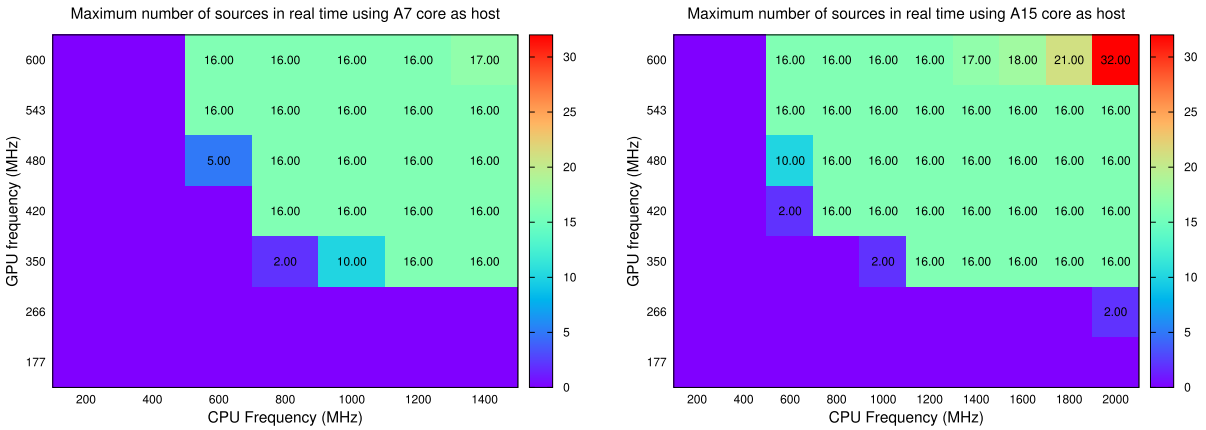

FIGURE 13. Maximum number of sources processed in real time using both kinds of CPU cores as host.

next value. The length of the steps depends on the device and spans 4 sources per core. The stepped behaviour arises from the fact that we have to synchronize all the work-items of every work-group for every sample and also because two threads per work-group perform additional computations, which involves several divergent branches on the code of the kernel. On the following experiments we use always `Dev0`, the fastest GPU device including 4 cores.

We leverage the possibility of regulating the frequency of the components of the platform to process as many sources as possible in real time with a minimal energy consumption. Obviously, by reducing the frequency of the CPU or GPU we increase the execution time of the algorithm and reduce the number of sources that we can process in real time. Fig. 12 shows the execution time required to process 8 sources. The type of CPU core employed as host of the OpenCL program has a small influence on the execution time and allows us to use lower frequencies to process the sources regardless of the CPU. The horizontal line defines the threshold imposed by the real-time constraint and shows that it is not possible to process 8 sources using the two lower frequencies of the GPU. However, even without using the largest frequencies, we can decrease the frequency of the CPU below 1,000 KHz and still be able to process 8 sources in real-time.

The type of CPU core used as host, as well as the frequency of the CPU and GPU cores, determine the maximum number of sources that can be processed in real-time. Fig. 13 allows us to analyze the effect of the frequencies on this number. The graphics only show the number of sources when it is possible to process at least one in real-time. Only by using medium or large frequencies we are able to process 256 samples of one or more sources in real-time. We can see also that the results are very similar when using both kinds of CPU cores. They only differ significantly with the highest CPU and GPU frequencies as we can then process 17 sources using a Cortex-A7 as host and up to 32 sources if we use a Cortex-A15 core.

We can also see that for most frequencies, the maximum number of sources that can be processed in real-time is 16. This value arises from the stepped behaviour of the time shown in Fig. 11. With the frequencies that allow us to process at least one source in real-time, execution time is almost constant and shorter than 5.80 ms with less than 16 sources and then goes up one step, overcoming the real-time threshold.

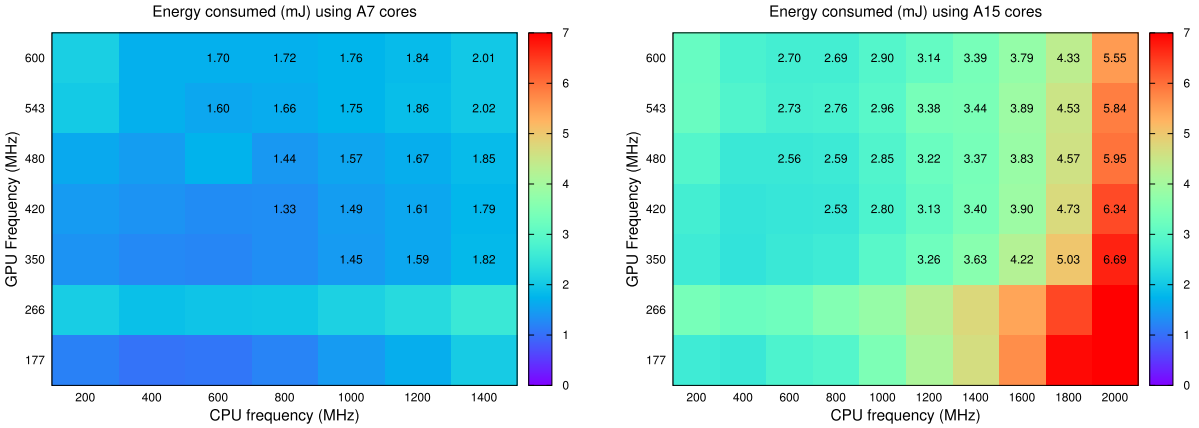

FIGURE 14. Energy consumption of the algorithm on the Exynos 5422 system-on chip to process 256 samples from 8 sources. Values are only shown when the sources can be processed in real-time.

Very often, energy consumption is a priority when designing applications for mobile devices, where saving battery life is of utmost importance. We obtain the energy consumption of our algorithm as the product of the measured power consumption of the whole platform while running it and the time taken by the algorithm to complete. We want to measure only the consumption of the application and so we subtract the power dissipated by the operating system processes, with the platform at the default frequencies.

We explore in Fig. 14 different configurations of the platform for minimizing the energy consumption needed to process 256 samples from 8 sources in real-time. We do not show values when it is not possible to render the sound sources in real-time. Note that using one Cortex-A7 core as host of the application is always more efficient in terms of energy consumption than using the more power-hungry Cortex-A15 cores. Besides, when using one A7 core, the energy consumption always increases both with the CPU and GPU frequencies. However, when using only one A15 core, the energy increases with the CPU frequency but decreases with the GPU frequency. That is, with this kind of core, increasing the frequency means more power dissipation, but it decreases the processing time so, when multiplying both factors to obtain the energy consumption of the algorithm the resultant value decreases.

Regardless of the CPU core employed as host, the most energy-efficient scenario is reached when decreasing the CPU frequency as much as possible while allowing the real-time processing of the sources. As for the GPU, when using a Cortex-A7 as host, the energy consumption is reduced by lowering the GPU frequency. Note, however, that when using a Cortex-A15, it is better to use the highest GPU frequency.

VII. CONCLUSION

This work proposes an efficient implementation of a spatial audio engine based on a parametric parallel filter bank for binaural synthesis. The proposed implementation is derived and optimized from the authors' previous work, where a low-order model based on a chain of SOS was shown to be an efficient approach for modeling and interpolating HRTF datasets. The limitations of conventional series-to-parallel conversion using partial-fraction-expansion have been presented to motivate the use of the developed parallel approach, which was designed to use efficiently the parallel computation resources found in modern mobile multimedia devices. The practical advantages of the model allows for an OpenCL-based implementation that can be run in a wide variety of platforms, including smartphones and tablets, contributing to its general portability.

The proposed OpenCL implementation has been tested on the GPU that is contained on a low-power and low-cost SoC aiming different objectives. On the one hand, we could render up to 32 sound sources in real-time without taking into account the energy consumption of the GPU. On the other hand, we combined two strategies to reduce energy consumption and save battery life in the device. Firstly, we used a low power-consuming Cortex-A7 core as host. Secondly, by lowering the working frequencies of both the CPU and GPU, we reduced the power required in the rendering. It was observed that the rendering of 16 sound sources produces the best trade-off in terms of sources rendered per watt. The results suggest that this value is achieved for different combinations of GPU and CPU frequencies. It is important to highlight that we are using GPU resources and freeing up CPU resources that can be exploited for other tasks, as this is an important consideration for mobile multimedia devices. Finally, comparing the architecture of the Mali-G76 MP12 GPU (Exynos 9820 SoC, released in 2019) with the Mali-T628 MP6 GPU (Exynos 5422 SoC, released in 2013) used in our experiments, we can conclude that our scalable implementation would be able to render in the modern GPU in real-time at least three times more sound sources. However, we cannot argue about the power required by carrying out such processing with the Mali-G76 GPU.

REFERENCES

[1] D. N. Zotkin, R. Duraiswami, and L. S. Davis, "Rendering localized spatial audio in a virtual auditory space," *IEEE Trans. Multimedia*, vol. 6, no. 4, pp. 553–564, Aug. 2004.

[2] M. Scarpino, *OpenCL in Action: How to Accelerate Graphics and Computations*. Shelter Island, NY, USA: Manning, 2012.

[3] B. R. Gaster, L. W. Howes, D. R. Kaeli, P. Mistry, and D. Schaa, *Heterogeneous Computing with OpenCL: Revised OpenCL 1.2 Edition*. San Mateo, CA, USA: Morgan Kaufmann, 2013.

[4] J. A. Belloch, A. Gonzalez, A. M. Vidal, and M. Cobos, "On the performance of multi-GPU-based expert systems for acoustic localization involving massive microphone arrays," *Expert Syst. Appl.*, vol. 42, no. 13, pp. 5607–5620, Aug. 2015.

[5] J. A. Belloch, A. Gonzalez, E. S. Quintana-Orti, M. Ferrer, and V. Valimaki, "GPU-based dynamic wave field synthesis using fractional delay filters and room compensation," *IEEE/ACM Trans. Audio, Speech, Lang. Process.*, vol. 25, no. 2, pp. 435–447, Feb. 2017.

[6] J. A. Belloch, A. Gonzalez, F. J. Martínez-Zaldívar, and A. M. Vidal, "Multichannel massive audio processing for a generalized crosstalk cancellation and equalization application using GPUs," *Integr. Comput.-Aided Eng.*, vol. 20, no. 2, pp. 169–182, Mar. 2013.

[7] F. L. Wightman and D. J. Kistler, "Headphone simulation of free-field listening. II: Psychophysical validation," *J. Acoust. Soc. Amer.*, vol. 85, no. 2, pp. 868–878, Feb. 1989.

[8] W. G. Gardner and K. D. Martin, "HRTF measurements of a KEMAR," *J. Acoust. Soc. Amer.*, vol. 97, no. 6, pp. 3907–3908, Jun. 1995.

[9] V. Pulkki, M.-V. Laitinen, and V. Sivonen, *HRTF Measurements With a Continuously Moving Loudspeaker and Swept Sines*. New York, NY, USA: Audio Engineering Society, May 2010.

[10] K. Iida, "Measurement method for HRTF," in *Head-Related Transfer Function and Acoustic Virtual Reality*. Singapore: Springer, 2019, pp. 149–156.

[11] V. R. Algazi, R. O. Duda, R. Duraiswami, N. A. Gumerov, and Z. Tang, "Approximating the head-related transfer function using simple geometric models of the head and torso," *J. Acoust. Soc. Amer.*, vol. 112, no. 5, pp. 2053–2064, Nov. 2002.

[12] E. A. Torres-Gallegos, F. Orduña-Bustamante, and F. Arámbula-Cosío, "Personalization of head-related transfer functions (HRTF) based on automatic photo-anthropometry and inference from a database," *Appl. Acoust.*, vol. 97, pp. 84–95, Oct. 2015.

[13] V. R. Algazi, R. O. Duda, D. M. Thompson, and C. Avendano, "The CIPIC HRTF database," in *Proc. IEEE Workshop Appl. Signal Process. Audio Acoust. (WASPAA)*, New Paltz, NY, USA, 2001, pp. 99–102.

[14] O. Warusfel. (2003). *Listen HRTF Database*. [Online]. Available: http://recherche.ircam.fr/equipes/salles/listen/

[15] Acoustics Research Institute (ARI). (2014). *ARI HRTF Database*. [Online]. Available: https://www.kfs.oeaw.ac.at/index.php?view=article&id=608&lang=en

[16] R. Bomhardt, M. de la Fuente Klein, and J. Fels, "A high-resolution head-related transfer function and three-dimensional ear model database," in *Proc. Meetings Acoust.*, 2016, vol. 29, no. 1, Art. no. 050002, doi: 10.1121/2.0000467.

[17] A. Kulkarni and H. S. Colburn, "Infinite-impulse-response models of the head-related transfer function," *J. Acoust. Soc. Amer.*, vol. 115, no. 4, pp. 1714–1728, Apr. 2004.

[18] J. Mackenzie, J. Huopaniemi, V. Valimaki, and I. Kale, "Low-order modeling of head-related transfer functions using balanced model truncation," *IEEE Signal Process. Lett.*, vol. 4, no. 2, pp. 39–41, Feb. 1997.

[19] G. Ramos and M. Cobos, "Parametric head-related transfer function modeling and interpolation for cost-efficient binaural sound applications," *J. Acoust. Soc. Amer.*, vol. 134, no. 3, pp. 1735–1738, Sep. 2013.

[20] G. Ramos, M. Cobos, B. Bank, and J. A. Belloch, "A parallel approach to HRTF approximation and interpolation based on a parametric filter model," *IEEE Signal Process. Lett.*, vol. 24, no. 10, pp. 1507–1511, Oct. 2017.

[21] U. Zolzer, *Digital Audio Signal Processing*, 2nd ed. Hoboken, NJ, USA: Wiley, 2008, pp. 115–168.

[22] V. Välimäki and J. Reiss, "All about audio equalization: Solutions and frontiers," *Appl. Sci.*, vol. 6, no. 5, p. 129, 2016.

[23] T. I. Laakso, V. Valimaki, M. Karjalainen, and U. K. Laine, "Splitting the unit delay [FIR/all pass filters design]," *IEEE Signal Process. Mag.*, vol. 13, no. 1, pp. 30–60, Jan. 1996.

[24] V. Valimaki and A. Haghparast, "Fractional delay filter design based on truncated Lagrange interpolation," *IEEE Signal Process. Lett.*, vol. 14, no. 11, pp. 816–819, Nov. 2007.

[25] H. Gamper, "Head-related transfer function interpolation in azimuth, elevation, and distance," *J. Acoust. Soc. Amer.*, vol. 134, no. 6, pp. EL547–EL553, Dec. 2013.

[26] R. Duraiswami, D. N. Zotkin, and N. A. Gumerov, "Interpolation and range extrapolation of HRTFs," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Montreal, QC, Canada, May 2004, pp. 45–48.

[27] A. V. Oppenheim, R. W. Schafer, and J. R. Bruck, *Discrete-time Signal Processing*. Upper Saddle River, NJ, USA: Prentice-Hall, 1975.

[28] R. A. Geiner and M. Schoessow, "Design aspects of graphic equalizers," *J. Audio Eng. Soc.*, vol. 31, no. 6, pp. 394–407, Jun. 1983.

[29] B. Bank and G. Ramos, "Improved pole positioning for parallel filters based on spectral smoothing and multiband warping," *IEEE Signal Process. Lett.*, vol. 18, no. 5, pp. 229–302, May 2011.

[30] S. Tassart, "Graphical equalization using interpolated filter banks," *J. Audio Eng. Soc.*, vol. 61, no. 5, pp. 263–279, May 2013.

[31] J. Ramo, V. Valimaki, and B. Bank, "High-precision parallel graphic equalizer," *IEEE/ACM Trans. Audio, Speech, Lang. Process.*, vol. 22, no. 12, pp. 1894–1904, Dec. 2014.

[32] W. Chen, "Performance of cascade and parallel IIR filters," *J. Audio Eng. Soc.*, vol. 44, no. 3, pp. 148–158, Mar. 1996.

[33] S. Barrachina, M. Barreda, S. Catalán, M. S. Dolz, G. Fabregat, R. Mayo, and E. S. Quintana-Ortí, "An integrated framework for power-performance analysis of parallel scientific workloads," in *Proc. 3rd Int. Conf. Smart Grids, Green Commun. IT Energy-Aware Technol. (ENERGY)*, Lisbon, Portugal, Mar. 2013, pp. 114–119.

[34] R. Gensh, A. Aalsaud, A. Rafief, F. Xia, A. Iliasov, A. Romanovsky, and A. Yakovlev, "Experiments with the Odroid-XU3 board," Dept. Comput. Sci., Newcastle Univ., Newcastle upon Tyne, U.K., Tech. Rep. CS-TR-1471, May 2015.

JOSE A. BELLOCH (Member, IEEE) received the degree in telecommunications engineering, the master's degree in parallel and distributed computing, and the Ph.D. degree in computer science from the Universitat Politècnica de València, Valencia, Spain, in 2007, 2010, and 2014, respectively. He was a Visiting Researcher with the Department of Signal Processing and Acoustics, Aalto University School of Electrical Engineering, Espoo, Finland, as a Predoctoral Researcher, in 2013, and as a Postdoctoral Researcher, in 2015. Since 2017, he has been an Assistant Professor with the Electronic Technology Department, Universidad Carlos III de Madrid, Madrid, Spain. He carried out an internship with the Department of Measurement and Information Systems, Budapest University of Technology and Economics, Budapest, Hungary. His work is focused on applying the new parallel architectures into signal processing algorithms. He has developed several real-time audio applications related with multichannel massive filtering, binaural sound, wave field synthesis systems, and sound source localization, using general purpose graphics processing units and ARM architectures. He was a recipient of the Extraordinary Ph.D. Thesis Award in recognition of his Ph.D. thesis. In 2016, he was honored with a Postdoctoral Fellowship of the Regional Government of Valencia in order to carry out research with the Universitat Jaume I de Castellón in collaboration with the Universidad Complutense de Madrid, Madrid.

GERMAN RAMOS (Member, IEEE) was born in Valencia, Spain. He received the M.S. degree in telecommunications engineering and the Ph.D. degree in digital audio signal processing from the Polytechnic University of Valencia, Spain, in 1997 and 2006, respectively. Since 1999, he has been a Lecturer on audio, DSP, acoustic, and electronics topics with the Polytechnic University of Valencia, being an Associate Professor, since 2012. Over the last 15 years, he has done regular consultancy for semiconductors and professional audio companies, with tens of products (software and hardware) on the market. His research interests are in the areas of digital filter design and implementation, optimized in-situ digital equalization of loudspeakers with low computational cost requirements, cross-over designs, psycho-acoustics, wave-field synthesis, and spatial audio. He has published more than 30 articles in these fields and holds international patent about low-computational cost audio filtering. He is a member of the AES.

JOSE M. BADIA was born in Valencia, Spain. He received the B.S. degree in computer science from the Polytechnic University of Valencia, Spain, in 1991, and the Ph.D. degree in computer science, in 1996. Since 1994, he has been a member of the Department of Computer Science and Engineering, University Jaume I, Castellón, Spain, where he was a Teaching Assistant, from 1994 to 1997, an Assistant Professor, until 2000, and has been an Associate Professor, since 2000. From 2007 to 2013, he was the Head of the Department of Computer Science and Engineering. He has published more than 40 articles in International conferences and journals. His main research interests include high performance computing for dense and sparse algebra, and power aware computing.

MAXIMO COBOS (Senior Member, IEEE) received the master's degree in telecommunications and the Ph.D. degree in telecommunications engineering from the Universitat Politècnica de València, Valencia, Spain, in 2007 and 2009, respectively. He completed his studies with honors under the University Faculty Training Program (FPU). In 2011, he joined the Universitat de Valencia, where he is currently an Associate Professor. His work is focused on the area of digital signal processing and machine learning for audio and multimedia applications, where he has authored/coauthored more than 90 technical articles in International journals and conferences. He is a full member of the Acoustical Society of America and a member of the Audio Signal Processing Technical Committee of the European Acoustics Association. He was a recipient of the Ericsson Best Ph.D. Thesis Award from the Spanish National Telecommunications Engineering Association. In 2010, he received a Campus de Excelencia postdoctoral fellowship to work at the Institute of Telecommunications and Multimedia Applications.