CORE
Metadata, citation and similar papers at core.ac.uk
Provided by Infoscience - École polytechnique fédérale de Lausanne

AN AUDIO VIRTUAL DSP FOR MULTIMEDIA FRAMEWORKS

Giorgio Zoia and Claudio Alberti

Integrated Systems Laboratory
Ecole Polytechnique Fédérale de Lausanne
giorgio.zoia@epfl.ch

ABSTRACT

The new MPEG-4 Audio standard provides two toolsets for synthetic Audio generation, Audio processing and multimedia content description called Structured Audio (SA) and BInary Format for Scenes (BIFS).

Moving from a systematic analysis of SA and from the implementation of an efficient SA decoder, this paper describes the design of a virtual DSP architecture able to exploit the data level parallelism contained in many typical audio processing algorithms. The proposed virtual DSP architecture shows good performance on general purpose platforms and can be easily adapted and optimized for parallel superscalar devices. The porting and results on a V-LIW DSP device confirm the effectiveness and flexibility of the approach, particularly suitable for standalone embedded solutions.

1. INTRODUCTION

In the last decade software digital audio signal processing has enormously evolved in functionality and acceptance among developers and content producers [1]. The reasons behind this explosion are various: first of all, the impressive increase in available computational power even in low price personal computers; then the great flexibility and ease in data manipulation provided by digital audio tools; finally the consequent migration of a musician's and content creator's education towards electronic and software oriented tools. The new MPEG-4 Audio standard can be considered a milestone in this evolutionary process, since it provides two toolsets to code advanced multimedia-oriented audio functionality, namely Structured Audio (SA) and BInary Format for Scenes (BIFS).

SA derives from years of academic research in software sound synthesis (SWSS) languages [1]; the SAOL (Structured Audio Orchestra Language, see [2]) programming language is similar to many of these SWSS tools but the whole SA toolset has been extended for multimedia and downloadable applications; the syntax of the language itself has become very close to C, with stronger object-oriented connotations and several stream-oriented extensions. SAOL maintains the typical execution scheme of its predecessors, with an internal *scheduler* (the fixed "main") and the possibility to define, inside its synthesis and processing functions, variables with different execution rates: initialization, control and audio rate. But unlike other similar tools, characterized by a block-based syntax, SA has a sample-by-sample (s-b-s) execution structure, that approaches it even more to C: this essentially means that syntax and semantics of statements and operators are defined for a single sample and not for a block of samples of length B_l

$$B_l = \text{srate/krate} \qquad (1)$$

where srate and krate are the sampling-rate and the control-rate respectively. If this makes possible a correct implementation of basic functions like recursive filters, on the other hand it introduces a relevant overhead in the case of an interpreted implementation of the language decoder[1], the most suitable for embedded real-time engines.

In MPEG-4, SA is surrounded at the system level by a higher-level language for scene description, BIFS; it is a language very similar in structure to VRML, but with an innovative and powerful audio-specific subtree called AudioBIFS [3]; functionality of AudioBIFS nodes ranges from simple mixing or delay until advanced spatialization schemes based on geometrical and perceptual information, so that a complete virtual audio environment can be described. Custom SAOL code can be linked to the scene through the AudioFX node, making of AudioBIFS an extremely powerful and flexible environment.

We present in the first half of this paper a new virtual DSP architecture designed starting from a platform independent profiling of the SAOL language; the DSP is able to exploit the block-based data level parallelism contained in many audio synthesis and processing algorithms, and to consistently reduce the SA execution overhead. In the second half of the paper we show how the DSP has been extended to include the complete AudioBIFS functionality, making it a complete audio engine for standardized multimedia description frameworks. The easy and effective porting of the virtual engine on a V-LIW DSP architecture validates the approach by showing how the task partitioning and the instruction set are suitable for modern mixed processor/superscalar-DSP solutions.

2. THE SAINT VIRTUAL DSP

The SAINT (for Structured Audio INTerpreter) SA decoder find its foundation in a systematic approach to the estimation of the SA decoding complexity, which must necessarily move from a time-dependent analysis and profiling of its typical applications. A new abstract method for measuring decoding complexity of normative Structured Audio programs has been developed [4] that provides platform independent metrics and that permits to carefully profile the execution of a program in function of time;

[1] In the MPEG-4 terminology the execution of a SAOL program is called decoding

in this way it is possible to characterize critical situations and to define a subset of the SAOL core library that is candidate for stronger optimization [5]. This new profiling method has been adopted in the MPEG-4 standard to define SA Levels of complexity and for SA Conformance testing [6].

2.1. Feedback analysis in Structured Audio

In parallel with the analysis of complexity, a second fundamental decoding issue has been considered: the possibility of a *block-by-block* (b-b-b) execution in SA, without altering the output of the normative s-b-s language specification.

Efficiency of a block-based execution over a sample based one has been previously proofed in literature [7]. In SA, what can prevent from executing b-b-b is the presence of an explicit feedback in the SAOL code. By explicit feedback we intend here a feedback programmed using more than one line of code, while an implicit one is for instance the case of the *iir* library function, where the feedback is hidden at a lower level, visible by the interpreter but not by the programmer. Explicit feedbacks have been detected by a simple graph analysis only in a few situations. The most obvious is when a new value is assigned to an audio variable after its first use. These cases have to be detected and treated in a special way, while the rest of the code can be executed on a possibly large b-b-b basis, typically with a block length of B_l as defined in (1). Of course, this can be done until the latency introduced by the block processing in the real-time synchronization of the complete MPEG-4 decoding process is tolerable (see [3]).

2.2. The SAINT architecture

Complexity analysis and feedback analysis demonstrated that in most cases an efficient implementation of the SA decoder can be obtained by the design of an extended multimedia DSP, based on a vectorial instruction set [8].

Since many multimedia-oriented superscalar devices are nowadays available, we imagined to design a virtual device with an instruction set that best matches the parallelism exploitable in many state-of-the-art DSPs, processors and multimedia processors [9,10].

With the SAINT approach, the complete decoding is divided in two independent layers: the decoder/compiler layer and the instruction layer. In this way the complete process is splitted into two separable parts, the initialization task and the real processing task; once this is accomplished, it is not difficult to run the first phase on a general purpose processor, and to execute the intensive processing possibly in the same CPU, but with the same effectiveness in a separate co-processor, single or even distributed; this is achieved through a simple sequence of method calls after a specific resource allocation that means allocation of the data space, method codes and their correct sequence. This architecture was conceived to target embedded solutions for standalone devices, which usually contain custom solutions based on low-power processors and DSPs.

The SAOL compiler is completed by a post-processor that optimizes some parts of the code, decomposes the standard library into core operations and finally performs a feedback detection analysis among the potential cases.

2.3. The SAINT DSP engine and its performance

The instruction set of the SAINT virtual DSP is essentially composed by *macroinstructions*, so called because they are conceived to possibly operate on blocks of samples. These macroinstructions are the methods that is directly executed by the ALU of the machine. All of them are defined in vectorial form, where the block of data on which the instruction is executed is defined by the values of two special registers in the machine, *start* and *end*. Unlike the case of e.g. the JAVA virtual machine [11], in SAINT there is no memory stack to work on, all the instructions directly operate on memory locations, and then they are defined with (usually) two or three addresses to load and store operators. This approach is more effective than the JAVA stack in our case, since in this case the coded information is transmitted by the MPEG-4 bitstream, and then there is no need to "compress" the compiled byte-code.

The scheduler works in direct contact with the ALU of the DSP, and then it must run on the same device where the code is being executed, in order to avoid useless communication between the two units. In fact, the scheduler can be seen as a master DSP unit working together with the processing DSP and executing a fixed, "hardwired" program of control and data postprocessing.

The virtual DSP architecture has been tested by several measurements with different versions of the decoder. The SAINT tool has been compiled on different platforms: we report here results on a PC with an Intel Pentium II at 400 MHz with 128 MB of RAM running Windows NT4. The software was compiled using BorlandC++ 5.02; optimization for speed was introduced. Many different groups of simulations have been conducted: results for a typical sequence of synthesis, based on wavetables with additional reverberation, are reported in Figure 1.

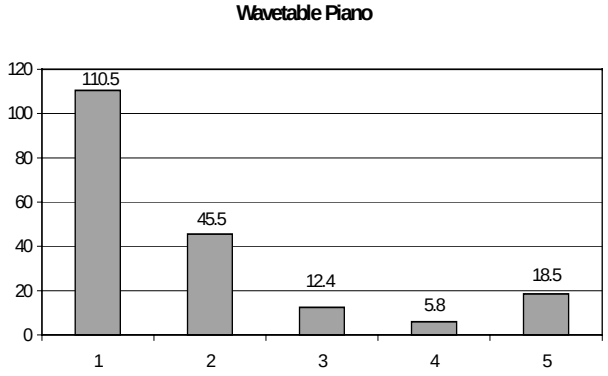

Figure 1. *Experimental results for different decoding approaches. Y-axis is time in seconds. The values of the five columns from left to right are the decoding time for: 1) the MPEG-4 SA reference software; 2) the SAINT decoder without any optimization; 3) the SAINT decoder with a block-by-block execution, when possible; 4) the SAINT decoder with the "Optivec" free downloadable vectorial libraries for Pentium; 5) the duration of the complete score file*

The mean polyphony of the score file, considering the effect of sustain, is approximately 3.5, the score duration is 18.5 seconds; the interpolation factor is 3. A considerable amount of static

optimization already gives a speed-up factor of 3 in comparison with the MPEG-4 reference software (columns 1 and 2). The b-b-b execution further introduces a speed-up factor of nearly 3, here with a block length of 441 (srate = 44100 and krate = 100). Finally, the introduction of handcrafted vectorial libraries on some basic functions, to replace macroinstructions written in C, shows how this approach can be effective: consider in fact that parallelism is exploited here only at the software level, while the vectorial instruction set can be optimized with a much greater efficiency on a truly parallel co-processor.

SAINT has also been compared with another available SA decoder based on a cross compiler approach (SAOL to C, see [12]), which requires of course a C compiler running *on* the target device while portablity of SAINT only requires a C compiler *for* the device. Compiling the C code in the same conditions and on different test sequences we have obtained results showing that SAINT is in the mean 20-30% slower than the cross-compiled approach. This figure is considered more than acceptable considering the weel-known drawback in speed between interpreted and compiled solutions.

3. MULTIMEDIA EXTENSIONS TO SAINT

An architecture and an instruction set conceived and optimized only for Structured Audio reveal several limitations when the device must be used for more general multimedia-oriented applications. Even if SAOL can potentially be used to describe any kind of algorithm, being a programming language and then a general audio coding scheme [13], many fundamental features of typical multimedia applications cannot be described with the necessary effectiveness.

Moving from the purpose of extending the MPEG-4 decoder to a complete AudioBIFS subtree processor, several extensions have been included in SAINT to support at best functionality of higher-level multimedia description languages.

3.1. From SAINT to BIFSAINT

The implementation of the complete AudioBIFS subtree requires the solution of non-trivial implementation challenges, first of all the correct synchronization between the Structured Audio built-in scheduler, which is active to process the AudioFX nodes, and the BIFS scene scheduler. An evaluation of practical implementation issues suggests that an effective solution can be the integration of the complete audio subtree into an extended *orchestra* (the overall SA program), where all the several nodes are transformed into SAOL functions and linked to the input/output buffers of the eventual AudioFX nodes in the correct sequence. It was then decided to "cross-compile" the AudioBIFS subtree to a sequence of extended SAOL functions and to proceed then to the execution of the complete audio scene using the SA scheduler as the overall mechanism for synchronization. This solution only requires a correct synchronization with the video part of the BIFS scene, whenever this should be present.

We mentioned earlier the necessity to extend the SAOL functionality; these extensions are necessary on both the instruction set and the memory management of the DSP, to deal essentially with more dynamical characteristics of BIFS, but also of other commonly used languages like VRML.

In SAOL the several instruments (i.e. the synthesis or processing functions) are statically defined at the beginning of the decoding process, when the bitstream header is received, and so are the routings among them; routings define the relationships among the input and output buffers of the different instruments. Only new instances of an existing instrument can be transmitted by streaming information.

In BIFS, as also in VRML, this is no more enough because routings among the nodes are dynamic and new nodes can be instantiated in the middle of the scene representation when new commands are received via the bitstream. Since in the proposed system the interaction between the parser/compiler and the execution engine is kept as far as possible separate, the virtual DSP must be made able to process streaming commands dealing with dynamic configurations of the input/output buffers, and above all with creation of new instruments. For instance, a new mixing *AudioMix* node could be instantiated, with a mixing matrix processing four inputs to produce two outputs, linked with sources and to target post-processing nodes.

A second class of extensions is necessary to support functionality provided by the more advanced nodes of BIFS. If some functions like mixers, delay lines and switchers are not difficult to translate into SAOL, on the other hand AudioBIFS provides even the possibility to spatialize monophonic sources using complex schemes such as room modelling or perceptual parameters. It is evident that in such cases some dedicated instructions are necessary to limit the overhead of the interpreted decoder to a minimum and to better optimize the transcoding into the intermediate format.

Finally, BIFS allows the presence of nodes working at different sampling rates, and lower rates must be converted to the higher ones at the most advanced point in the processing tree. This is not possible in standard SAOL, where all the instruments must work at the same audio rate. It is then necessary to extend the scheduler with the capability to treat different subgroups of instruments (i.e. different orchestras) that are combined through asynchronous sampling-rate converters at the correct point.

3.2. The ThreeDSPACE Audio framework

The concept and design of SAINT and of its extended BIFS version are inserted in the context of the ThreeDSPACE project. ThreeDSPACE is a mixed academic/industrial research project aiming at implementing advanced 3-D audio processing and rendering methods, compatible with the MPEG-4 AudioBIFS specification. The chosen technology for spatial processing in ThreeDSPACE is wave field synthesis (WFS, [14]). To apply this technology loudspeaker stripes, containing a high number of separate channels, must be mounted directly to walls. The signals are generated by dedicated signal processors, which need suitably encoded sound-field representations as inputs. A spherical wave coming from a virtual source behind the array, or the extrapolated wavefront of a source in front of the array can be reproduced using the WFS solution [14].

In this framework, SAINT is used as signal generator and preprocessor. The decoding of the AudioBIFS scene, the only standardized format able to support the necessary information for WFS rendering, is not carried on until the sound composition in the virtual DSP itself. Instead, monophonic audio sources and side information for control are sent to the output of the first

processing stage, and active loudspeaker stripes will produce the necessary number of channels (64 per stripe) in order to generate the holophonic sound field. The overall system and data flow of the SAINT decoder used in ThreeDSPACE is represented in Figure 2.

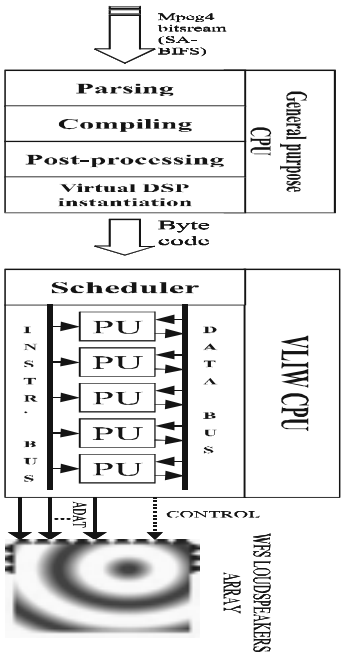

Figure 2 *The data flow through the ThreeDSPACE audio framework. Mono channels and control information are processed by active arrays of loudspeakers to produce suitable WFS multiple channels.*

The entire system can be splitted into three main stages. The first one, dealing with the parsing and compilation of the MPEG-4 bitstream, has been implemented on a general purpose CPU platform. The second stage contains the virtual DSP and has been ported on a Philips TriMedia multimedia DSP, a V-LIW architecture. The processor runs at 100MHz and its V-LIW instruction set allows up to five simultaneous operations to be issued. The proposed block-by-block approach shows its potential exploiting this feature, reaching average values of 4 processing units busy per clock cycle in some benchmarks. Moreover, since the feedback analyzer and the virtual DSP execution engine a-priori avoid the presence of feedback loops, a significant speed-up has been measured declaring special pointers not referring to overlapping memory locations. The aggressive exploitation of these features lets the virtual DSP run at a speed comparable to the one obtained on a 200MHz general purpose CPU (Pentium MMX), with output audio formatting in addition.
Monophonic output channels are then sent in ADAT format to the arrays of loudspeakers, where WFS is implemented with the support of side control information delivered by the BIFSAINT decoder on a further channel.

4. CONCLUSION

We have presented in this paper the design and implementation of a virtual DSP engine dedicated to exploitation in multimedia environments like those proposed by the new MPEG-4 standard. Additional information about our work on SAINT and ThreeDSPACE can be found at http://lsiwww.epfl.ch .

5. REFERENCES

[1] Roads, C., 1996. "The Computer Music Tutorial". Cambridge, MA: MIT Press.

[2] Scheirer E.D., B. L. Vercoe: "SAOL: The MPEG-4 Structured Audio Orchestra Language." Computer Music Journal 23 (2) : pp. 23-35, 1999.

[3] Scheirer, E., J. Huopaniemi and R. Väänänen "AudioBIFS: The MPEG-4 Standard for Effects Processing." Proc. DAFX98 Workshop on Digital Audio Effects, Barcelona, Nov. 1998

[4] Zoia, G. "A method for Complexity Measurements in Structured Audio". ISO/IEC JTC1/SC29/WG11 (MPEG98) document M3602, Dublin - July 1998.

[5] Zoia G., C. Alberti "A Virtual DSP Architecture for MPEG-4 Structured Audio", Proceedings of the COST-G6 Conference on Digital Audio Effects - DAFx-00, Verona, Italy, December 2000.

[6] ISO/IEC JTC1/SC29/WG11 (MPEG99) document N3067-sub3. "Information Technology - Coding of Audio-Visual objects. Part 4: Conformance. Subpart 3: Audio Conformance". MPEG-4 Audio International Standard.

[7] Dannenberg, R. B., N. Thompson: "Real-Time Software Synthesis on Superscalar Architectures". Computer Music Journal 21 (3) : pp. 83-94, 1997.

[8] Zoia G., C. Alberti "An Efficient Block-Based Interpreter for MPEG-4 Structured Audio", Proceedings of the IEEE International Symposium on Circuits and Systems - ISCAS 2000, Geneva, Switzerland, May 2000.

[9] Espasa R., M. Valero: "Exploiting Instruction- and Data-Level Parallelism". IEEE Micro, September - October 1997 : 20-27.

[10] Flynn, M. J.: "Computer Architecture: Pipelined and Parallel Processor Design". Sudbury, MA: Jones and Bartlett Publishers, 1995.

[11] Lindholm T. and F. Yellin "The JAVA Virtual Machine Specification". 2nd Edition (JAVA series), Addison Wesley, 1999.

[12] Lazzaro J. and J. Wavrzynek "MPEG-4 Structured Audio: developer tools" CS Division, UC Berkeley, http://www.cs.berkeley.edu/~lazzaro/sa/index.html

[13] Scheirer E., Y. Kim "Generalized Audio Coding with MPEG-4 Structured Audio". AES 17th conference on High Quality Audio Coding.

[14] Boone M., E. Verheijen, G. Jansen "Virtual Reality by Sound Reproduction Based on Wave Field Synthesis" Proceedings of the 100th AES Convention, Copenhagen, Denmark, 1996.