ZERO-DELAY LARGE SIGNAL CONVOLUTION USING MULTIPLE PROCESSOR ARCHITECTURES

Nicholas Jillings[1], *Joshua D. Reiss*[2], *Ryan Stables*[1]

[1] Digital Media Technology Lab, Birmingham City University, Birmingham UK
nicholas.jillings@bcu.ac.uk, ryan.stables@bcu.ac.uk
[2] Centre for Digital Music, Queen Mary University of London, London, UK
joshua.reiss@eecs.qmul.ac.uk

ABSTRACT

Zero latency convolution typically uses the Direct Form approach, requiring a large amount of computational resources for every additional sample in the impulse response. A number of methods have been developed to reduce the computational cost of very large signal convolution. However these all introduce latency into the system. In some scenarios this is not acceptable and must be removed. Modern computer systems hold multiple processor architectures, with their own strengths and weaknesses for the purpose of convolution. This paper shows how correctly combining these processors can lead to a powerful system which can be deployed for real-time, zero-latency large signal convolution.

Index Terms— Convolution, GPU, Processor, Direct Form

1. INTRODUCTION

Convolution is used to perform a range of computational processes in audio production, including the application of digital filters, or reverberation effects [1]. This is traditionally implemented as Direct Form (Eq. 1), but its computational cost increases dramatically as the impulse length increases [2], illustrated in Figure 1. Several methods have been proposed to reduce the per-sample computational cost. The major breakthroughs are multi-block uniform partitions (MBUP) [3], multi-block non-uniform partitions (MBNUP) [4] and the Gardner method [5]. These algorithms generally trade computational efficiency for system latency or complexity. Ideally however, a desirable system will reduce computational cost without introducing system latency.

$$y[n] = x[n] * h[n] = \sum_{m=0}^{M-1} x[n]h[m-n] \tag{1}$$

The convolution process is traditionally executed on the computers' CPU [6], where vector instruction sets can improve performance [7]. Modern GPU processors however, can also perform general purpose computing work in addition to image and video rendering [8]. Many aspects of computational research have been accelerated by introducing GPUs, such as image processing [9], machine learning [10, 11] and audio processing [12], offloading work from CPUs. This however, is only useful for certain scenarios as system latency is introduced [13, 14].

Computational tasks on the GPU only benefit if there is sufficient parallelism to *saturate* a GPU processor [13, 14, 15]. GPUs can have significant latencies waiting for shared resources, such as memory access. The host interface bus and GPU architecture can add extra limitations and latencies [14]. Tasks which do not meet these criteria may under-perform compared to a CPU [14, 15].

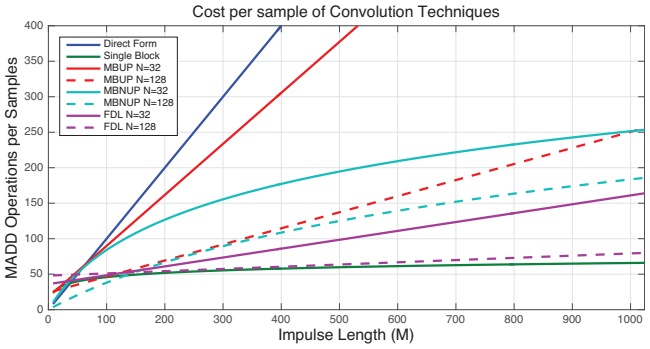

Figure 1: Operation counts for a number of convolution algorithms with increasing partition size, as outlined in [16]. These calculations assumed an FFT factor of $k = 2.0$.

Section 2 identifies a number of key algorithms for performing convolution. The test methodology is then outlined in section 3, with results presented in section 4. Finally, section 5 proposes a novel system for zero-delay convolution, by combining both CPU and GPU architectures.

2. CONVOLUTION METHODS

Direct Form convolution (as shown in Eq. 1) requires M multiplications and $M-1$ additions per sample, where M is the length of the impulse response. This quickly becomes expensive to implement on modern hardware as shown in Figure 1.

Early methods to lower the cost, without inducing large latencies or resampling [17], involve partitioning a long impulse response into smaller individual sub-filters [3, 18, 19]. The impulse signal h is divided into a set of sub-filters h_k, with lengths equal to the partition size N, where $1 \leq N \leq M$. Here, h_k is defined as a subset of h, where $h[n] \in h_k$, and $Nk \leq n < N(k+1)$. The same process is applied to the input signal x to give two sets of vectors: $[h_0, h_1, ..., h_j]$ and $[x_0, x_1, ..., x_j]$. These frames are then processed in the frequency domain, as shown in Eq. 2, which is typically more efficient than the Direct Form, but requires buffering.

$$y_j = \text{IFFT}\left(\sum_{k=0}^{K} H_k \text{FFT}\left(x_{j-k}\right)\right) \tag{2}$$

978-1-5386-1631-4/17/$31.00 ©2017 IEEE

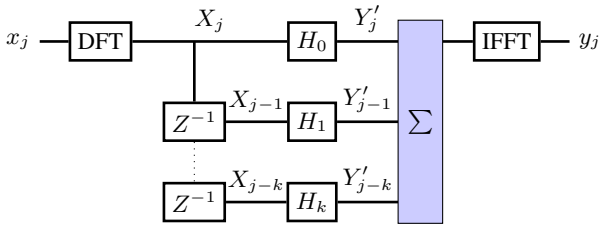

Figure 2: Block diagram of the frequency delay line

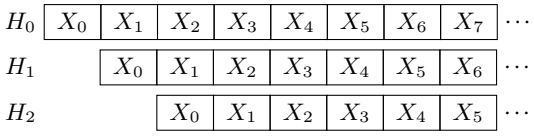

Figure 3: Uniform Partition with three sub-partitions [3].

Shorter partitions require fewer samples to buffer, decreasing the latency, but increasing computation as the smaller FFT sizes are less efficient (see Figure 1 - MBUP). The resulting frames are then summed and passed through an IFFT to get y_j. This function can be further optimised by performing the FFT and storing the frequency domain information, giving it the term Frequency-domain Delay Line (FDL) [6, 20], shown in Figure 2.

Partitioning the signal in this way simplifies parallel processing [20, 21] since certain computations can be performed ahead of time. In Eq. 2, only a convolution with x_j will require a delay, as x_{j-1} and x_{j-2} are already stored in memory. Figure 3 outlines the process scheduling for a uniform partition. In practice, all scheduling happens at once and the intermediary result is stored to ensure real-time execution. This also has a uniform execution profile since the same number of frames are processed and every frame is the same size.

Nonuniform partition scheduling (illustrated in Figure 4) benefits from the low latency of small partitions and the efficiency of large partitions [4]. By doubling the partition size every time the cost drops, the same signal length can be covered in fewer partitions. Therefore the system latency is equal to the size of the first partition. However, the computational load is nonuniform, meaning a new frame may not need to be processed by every sub-filter every time. But each sub-filter must still be processed within N_k/f_s seconds, where N_k is the length of the k^{th} partition. By processing small partitions first, it is then feasible to perform a Direct Form convolution for H_0 and a nonuniform method for the rest [5], as shown in Figure 5.

3. METHODOLOGY

Three convolution algorithms (Uniform Partitions [3], Nonuniform Partitions [4] and Gardner [5]) were tested on a set of mobile, consumer-level processors (Tables 1 and 2), spanning several CPU and GPU generations and architectures. On the CPUs, the algorithms were compiled as C functions using available instruction extensions (SSE, AVX and FMA). For the GPUs, the algorithms used OpenCL (AMD, Intel) and CUDA (nVidia) compiled kernels. The processes used single precision floating points as consumer GPUs do not have hardware for double precision calculations. The algorithms were executed off-line to obtain the maximum throughput of the processors for a wide range of impulse response lengths ($32 \leq M \leq 2^{24}$) and partition sizes ($32 \leq N \leq M$). Each combination of M and N executed for 10 seconds. During this time period, the number of samples processed is observed, providing the throughput as *samples per second*. If this is greater than the sample rate f_s then the system can be considered real-time.

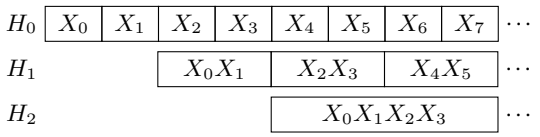

Figure 4: Nonuniform Partition with three sub-partitions[4].

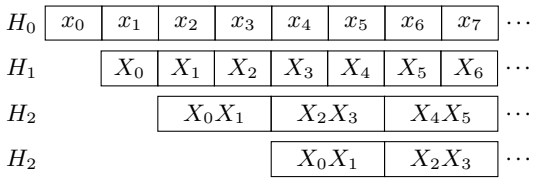

Figure 5: *Gardner* technique with four sub-partitions [5].

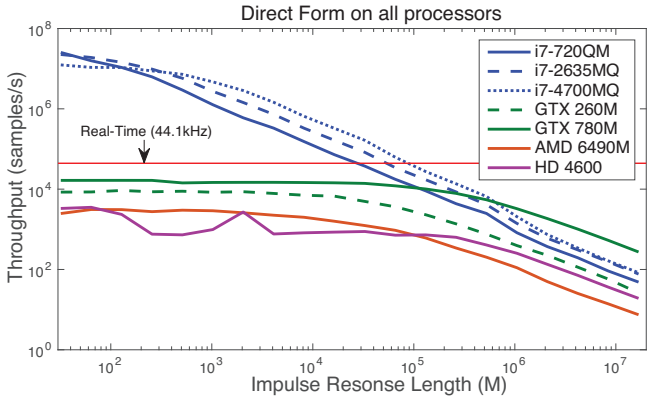

Figure 6: Throughput of all processors operating the Direct Form.

4. INDIVIDUAL PROCESSOR PERFORMANCE

4.1. Direct Form

The performance of the Direct Form algorithm for all processors is shown in Figure 6. The red line is the minimum throughput for a system to be real-time. Above this, the processor can be used in a real-time system. The figure shows only the tested CPUs are able to meet this constraint. CPUs execute this with a multiply-accumulation loop, giving a nearly linear response; doubling M halves the throughput. Generational differences vary the maximum signal that can be computed. Here, the CPUs were affected by the cache [22], where each experienced a drop in performance when M exceeded half the size of the L3 cache.

None of the tested GPUs could process the signal in real-time. The discrete GPUs failed because the high data rates of the PCI-E

Test Machine	Alienware M15x	MacBook Pro 2011	Alienware 17
Package Name	Intel Core i7-720QM	Intel Core i7-2635QM	Intel Core i7-4700MQ
Code Name	Nehalem Clarksfield	Sandy Bridge	Haswell-MB
Cores (Phy/Vir)	4 (8)	4 (8)	4 (8)
Base Clock (Turbo)	1.6GHz (2.8GHz)	2.0GHz (2.9GHz)	2.4GHz (3.4GHz)
Memory Capacity	8GB	4GB	16GB
L1/L2/L3 Cache	32KB / 256KB / 6MB	32KB / 256KB / 6MB	32KB / 256KB / 6MB
Vector Instructions	SSE4.2	SSE 4.2, AVX	SSE4.2, AVX, AVX2, FMA

Table 1: Specifications of the CPU packages tested

Test Machine	Alienware M15x	MacBook Pro 2011	Alienware 17	
Package Name	nVidia GTX 260M	AMD Radeon HD 6490M	Intel HD 4600	nVidia GTX 780M
System Interface	PCI-E v1 x16	PCI-E v2 x16	SoC Interconnect	PCI-E v3 x16
Package RAM	1024MB	256MB	0MB	4096MB
Language	CUDA 1.1	OpenCL 1.2	OpenCL 1.2	CUDA 3.0

Table 2: Specifications of the GPU packages tested

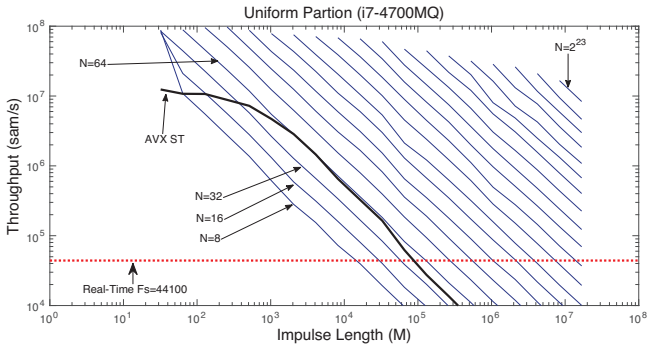

Figure 7: Intel i7-4700MQ operating uniform-partition test of various partition sizes (N) compared with the same processor executing the Direct Form using AVX (black line)

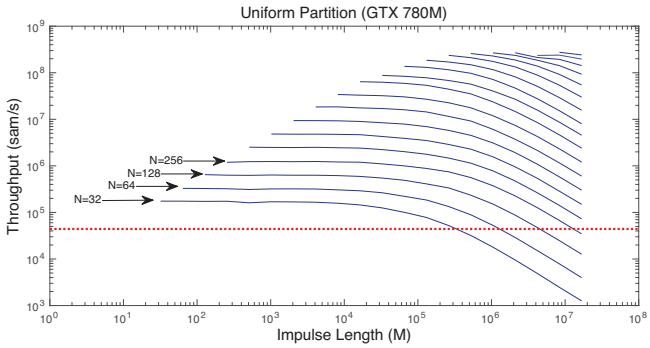

Figure 8: GTX 780M operating FDL test of various partition sizes (N). The red line is 44100 samples/second, above this line the system is real-time.

interface are only achieved when large frames of data are transferred infrequently [14]. The integrated Intel HD 4600 GPU was expected to perform well since it uses the same memory pathways as the CPU. However the processor is significantly less powerful resulting in longer execution times.

4.2. Uniform Partitions

The uniform partition is the simplest way to efficiently parallelise the convolution algorithm [3]. Table 3 gives the number of frames processed per second for each processor, where multiplying the frames per second with the partition size N gives the throughput value. For the CPUs, if the partition size doubles the throughput roughly doubles, since the number of processed frames is fairly consistent. Table 3 shows that increasing N by a factor of 256 (64 to 16,384) only drops the number of frames by an average factor of 1.73. Figure 7 shows an example of this using the i7-4700MQ CPU.

The discrete GPUs could all process real-time, even with a small N, due to the increased efficiency from transferring frames of audio less frequently. When the number of filter coefficients is less than the number of workers, the processor is under-utilised. In this state, adding more work does not decrease the processor throughput. Figure 8 provides an example of this using the GTX 780M, however all other tested GPUs exhibit similar behaviour. When the GPUs are fully saturated they exhibit similar performance gains to the CPUs, where doubling N does not significantly reduce the frames processed per second, shown in Table 3 . The integrated HD 4600 GPU had instability in testing, likely caused by the CPU consuming resources.

4.3. Nonuniform partitions

Nonuniform partitioning was the most efficient method of operation on the CPU as shown in Table 4, compared with Table 3. The nonuniform method can run with zero-delay by operating the first partition in the time domain, rather than the frequency domain [5]. It is not possible to execute the non-uniform method on a GPU due to the varying partition sizes. The GPUs execute the FFT as a series of identical transforms, processing several vectors at once (batch execution). It would be inefficient to use the GPU for several, non-uniform transforms. Equally it would be complex to map the shared memory of non-uniform blocks onto fixed worker pool sizes.

N	i7-720Q	i7-2635QM	i7-4700MQ	GTX 260M	GTX 780M	Intel HD 4600	AMD HD 6490M
64	796	756	1,288	1,569	3,471	811	455
256	804	616	1,395	1,593	3,623	1,041	618
1024	696	546	1,265	1,346	3,348	942	543
16384	493	338	969	438	2,193	414	255

Table 3: Audio frames processed per second operating the uniform-partition convolution for various partition sizes (N) with M=65,536.

N_0	i7-720Q	i7-2635QM	i7-4700MQ
64	31,231	27,129	63,531
256	9,676	7,953	20,151
1024	3,156	2,540	6,566

Table 4: Audio frames processed per second operating the nonuniform partition convolution for various initial partition sizes (N_0) with M=65,536.

5. MULTI-PROCESSOR CONVOLUTION

To achieve large-signal convolution with zero-delay in real-time multiple processors can be used. A CPU and GPU topology can achieve this using a partitioned process with Direct Form as the first partition. The results show the CPU is the only processor type capable of performing the Direct Form in real-time (Figure 6). The results also show the GPU outperforms the CPU in the partitioned algorithms (Table 3). A system which achieves both zero-latency for large signal convolution using both the CPU and GPU in the same processing chain is proposed.

In this section M_C and M_G are the impulse lengths the CPU and GPU can process individually ($M = M_C + M_G$). N_0 is the length of the first partition, executed on the CPU and N_G is the length of the partitions on the GPU.

5.1. Maximize convolution size

The first architecture outlines how to maximize the convolution size that can be processed on a system whilst meeting real-time constraints. The first filter partition must be processed using the Direct Form on the CPU. All CPUs have a point where the Direct Form becomes more expensive than the frequency domain convolution [5]. This point is the 'cross over' between the two methods. To allow a partition-based system to operate ahead of schedule, guaranteeing real-time performance [5], the Direct Form computation must be double this value.

The remaining CPU cores can perform the FDL (see Figure 2) or the nonuniform partition to cover more of the signal. Each core can operate larger partition sizes than the previous since its processing time can be masked by the combined length of the previous cores. For instance, if two cores process a total of 1,024 samples then the third core can operate at a partition size of up to 512 samples. The GPU then processes the remainder of the signal using an FDL, the most efficient type investigated. The GPU can select a partition size up to half of the entire impulse length covered by the CPU ($N_G = 0.5M_C$). Figure 9 gives a simplified overview of the communication and synchronising between the processors.

As an example, the Alienware 17 platform (see Tables 3 and 4) can process 133,777,344 samples per second using this method. One CPU core operates a Direct Form of $N_0 = 128$, its tested cross-over point. The next core operates an nonuniform partition up to the tested $M = 2^{24}$ with a first partition size of $N_0/2 = 64$. This gives the CPU a theoretical processing length of $M_C = 128 + 2^{24}$. The HD 4600 processor could also operate high volumes, however it will reduce CPU throughput since it is on-chip, so is unused. The GTX 780M could use any tested partition rate, limited only by the amount of RAM on-board (4096MB, or $M_G = 1.17 \times 10^8$). The total number of samples then equals $M = M_C + M_G = 128 + 2^{24} + 1.17 \times 10^8$ which equals 3,033.5s (at $f_s = 44100$).

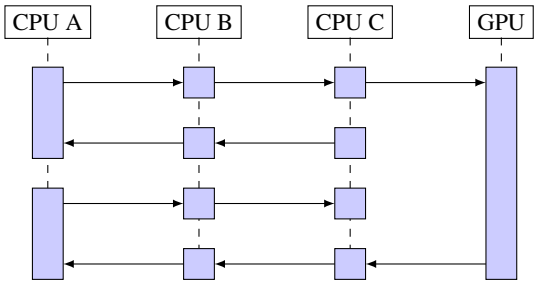

Figure 9: Example cycle of the multiprocessor scheduling. CPU A performs the Direct Form with CPUs B and C operating the frequency delay line at half frame size. The combined delays allows the GPU to process four times the frame length per invocation

5.2. Minimize CPU load

The CPU is a heavily shared resource, having to maintain other software tasks on a modern machine. It is therefore undesirable to operate the CPU at 100%. As with the previous method, one CPU core must operate the Direct Form at double the cross-over point. The GPU can then perform an FDL with a partition size up to $N_G = M_C/2$. If more samples are needed, the CPU can perform limited FDLs to extend the total length of time.

To provide an example, the Alienware 17 platform's CPU can operate a very small fraction of the total impulse response. The CPU must still perform a Direct Form at $N_0 = 128$, the calculated efficiency cross-over. Therefore $M_C = N_0$ since it only performs this task. At this level the GTX 780M can actually perform a very high throughput (see Table 2), achieving $M_G = 2^{21}$. In this case the total convolution length M is 2,097,280 samples (47.55s at $f_s = 44100$). This is significantly shorter than the maximum length, but the theoretical CPU operating time is 35.532ns per sample or 0.157% of maximal capacity.

6. CONCLUSION

This paper has introduced a novel method for processing large convolution kernels in real-time with zero system latency. It has outlined the existing methods for performing low-latency convolution

on discrete signals. These methods were executed on a number of modern system architectures and their effectiveness were evaluated. Any weaknesses were identified, such as CPU code efficiencies and GPU memory communication latencies. The proposed method combines the CPU and GPU in a processing chain, outlining a method for maximum performance using the CPU and GPU at full capacity. It also shows the GPU can offload the majority of this work, allowing for large signal convolution of over 2 million samples with a theoretical CPU usage of under 1%.

7. REFERENCES

[1] V. Valimaki, J. D. Parker, L. Savioja, J. O. Smith, and J. S. Abel, "Fifty years of artificial reverberation," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 5, pp. 1421–1448, July 2012.

[2] J. O. Smith III, *Mathematics of the Discrete Fourier Transform (DFT), with Audio Applications*, 2nd ed. http://www.w3k.org/books/: W3k Publishing, 2007, ISBN 978-0-9745607-4-8.

[3] C. Burrus, "Block realization of digital filters," *IEEE Transactions on Audio and Electroacoustics*, vol. 20, no. 4, pp. 230–235, Oct 1972.

[4] G. P. M. Egelmeers and P. C. W. Sommen, "A new method for efficient convolution in frequency domain by nonuniform partitioning for adaptive filtering," *IEEE Transactions on Signal Processing*, vol. 44, no. 12, pp. 3123–3129, Dec 1996.

[5] W. G. Gardner, "Efficient convolution without input/output delay," in *Proc. of the 97th Audio Engineering Society Convention*. San Francisco, CA, USA: Audio Engineering Society, Nov. 1994.

[6] E. Battenberg and R. Avizienis, "Implementing real-time partitioned convolution algorithms on conventional operating systems," in *Proceedings of the 14th International Conference on Digital Audio Effects. Paris, France*, 2011.

[7] T. Kite, "IIR filters for audio test and measurement: Design, implementation, and optimization," in *Audio Engineering Society Convention 137*, Oct 2014. [Online]. Available: http://www.aes.org/e-lib/browse.cfm?elib=17507

[8] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krger, A. E. Lefohn, and T. J. Purcell, "A survey of general-purpose computation on graphics hardware," *Computer Graphics Forum*, vol. 26, no. 1, pp. 80–113, 2007. [Online]. Available: http://dx.doi.org/10.1111/j.1467-8659.2007.01012.x

[9] J. Ao, S. Mitra, and B. Nutter, "Fast and efficient lossless image compression based on cuda parallel wavelet tree encoding," in *2014 Southwest Symposium on Image Analysis and Interpretation*, April 2014, pp. 21–24.

[10] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, June 2012, pp. 3642–3649.

[11] D. Steinkraus, I. Buck, and P. Y. Simard, "Using GPUs for machine learning algorithms," in *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*, Aug 2005, pp. 1115–1120 Vol. 2.

[12] N. Tsingos, W. Jiang, and I. Williams, "Using programmable graphics hardware for acoustics and audio rendering," *J. Audio Eng. Soc*, vol. 59, no. 9, pp. 628–646, 2011. [Online]. Available: http://www.aes.org/e-lib/browse.cfm?elib=15979

[13] L. Savioja, V. Välimäki, and J. O. Smith, "Audio signal processing using graphics processing units," *JAES*, vol. 59, no. 1/2, pp. 3–19, Jan/Feb 2011. [Online]. Available: http://www.aes.org/e-lib/browse.cfm?elib=15772

[14] N. Jillings and Y. Wang, "CUDA accelerated audio digital signal processing for real-time algorithms," in *Proc. of the 137th Audio Engineering Society Convention*. Los Angeles, CA, USA: Audio Engineering Society, Oct. 2014.

[15] R. Vuduc, A. Chandramowlishwaran, J. Choi, M. Guney, and A. Shringarpure, "On the limits of GPU acceleration," in *Proceedings of the 2nd USENIX conference on Hot topics in parallelism*, vol. 13. USENIX Association, 2010.

[16] G. Garcia, "Optimal filter partition for efficient convolution with short input/output delay," in *Proc. of the 113th Audio Engineering Society Convention*. Los Angeles, CA, USA: Audio Engineering Society, Oct. 2002.

[17] D. A. Burgess, "Techniques for low cost spatial audio," in *Proceedings of the 5th Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '92. New York, NY, USA: ACM, 1992, pp. 53–59. [Online]. Available: http://doi.acm.org/10.1145/142621.142628

[18] A. Torger and A. Farina, "Real-time partitioned convolution for ambiophonics surround sound," in *Proceedings of the 2001 IEEE Workshop on the Applications of Signal Processing to Audio and Acoustics (Cat. No.01TH8575)*, 2001, pp. 195–198.

[19] C. Mller-Tomfelde, "Low-latency convolution for real-time applications," in *Audio Engineering Society Conference: 16th International Conference: Spatial Sound Reproduction*, Mar 1999.

[20] F. Wefers and J. Berg, "High-performance real-time FIR-filtering using fast convolution on graphics hardware," in *Proc. of the 13th Conference on Digital Audio Effects*, Grax, Austria, Sep. 2010.

[21] J. A. Belloch, A. Gonzalez, F. J. Martínez-Zaldívar, and A. M. Vidal, "Real-time massive convolution for audio applications on gpu," *The Journal of Supercomputing*, vol. 58, no. 3, pp. 449–457, 2011. [Online]. Available: http://dx.doi.org/10.1007/s11227-011-0610-8

[22] T. Ilmonen and T. Lokki, "Extreme filters-cache-efficient implementation of long iir and fir filters," *IEEE Signal Processing Letters*, vol. 13, no. 7, pp. 401–404, July 2006.