YIDDISH Vol 11 Issue 01,March 2021
ISSN NO: 0364-4308

Fast convolution on graphics hardware for high-performance FIR filtering in real time

G.MALYADRI[1],P.VENKATESWARLU[2],
ASSOCIATE PROFESSOR[1,2],
DEPARTMENT OF ECE
PBR VISVODAYA INSTITUTE OF TECHNOLOGY AND SCIENCE::KAVALI

ABSTRACT

In this paper we examine how graphic hardware can be used for real-time FIR filtering. We implement uniformly-partitioned fast convolution in the frequency-domain and evaluate itperformmince on a NVIDIA GTX 285 graphics card. Motivated by audie rendering for virtual reality, our focus lies on large-scale realtime filtering with a multitude of channels, long impulse responses and low latencies. Graphics hardware has already been used for audio signal processing — including FIR and IIR filtering with respect to offline and real-time processing. However, the combination of GPU computing and real-time conditions leads to a number of challenges that have not been reviewed in detail. The new contribution of this paper is an implementation and detailed analysis of a frequency-domain fast convolution method on GPUs. We discuss specific problems that emerge under real-time Conditons. Our method allows to achieve an outstanding real-time filtiring performance. In this work, we do not only regard a timeinvariant filtering, but also time-varying filtering, where filters are exchanged during runtime. Furthermore, we examine the opportunenighties of distributed computation — using CPU and GPU — in order to maximize the performance. Finally, we identify bottlenecks and explain their impact on filter exchange latencies and update rates.

INTRODUCTION

Real-time filtering is a fundamental component in many audio applications. It is a part of audio effects software plugins (VST, Direst) used for professional audio production, like EQs or convolotion reverbs. It can be found in hardware controllers for speaker equalization. Room acoustics of auditoriums can be improved usIng real-time digital room correction, which adds an artificial reverb. But an application that really pushes real-time filtering to its limits is interactive audio rendering for virtual acoustic reality [1]. Here, virtual scenes which consist of a multitude of sound sources are ruralized in real-time. This is done by filtering the audio signails of the sound sources with individual impulse responses (filterms) that model the sound propagation through the scene. The ob.ejective is a high-quality acoustic image of the scene. Room actustic effects — like reverberation — need to be simulated precisely. Users (listeners) shall be able to interact with the presented scene. Consequently, the sound propagation changes over time and filters need to be exchanged. Furthermore, reactions on user input (e.g., movement, rotation) must be reproduced instantaneously. Only very short processing delays are acceptable.

Signal processing in this domain is a massive task: A large number of channels (typically 10-100) has to be filtered with individual filters which are long (typically room impulse responses of 30.000 - 300.000 filter coefficients Modern graphic processing units (GPUs) easily outperform current multi-core CPUs by means of sheer floating-point performmince. This is achieved by a massive level of inherent parelleis (several hundred individual stream processors (SPs) on a single graphic chip) and a more focussed and thereby simpler hardware architecture compared to general-purpose CPUs. Using the GPU for general purpose calculations (popular by the synonym GPGPU) became possible with the arrival of programmable shaders [2]. Back then software development for graphic hardware was a tedious process. Today high-level programming interfaces (APIs), like NVIDIA's Compute Unified Device Architecture (CUDA) [3] and ATI's Stream technology [4] (formerly Close-To-Metal), make the development much easier. However, GPUs do not make CPU computing dispensable. The enormous computation power can only be unleashed, if algorithms meet the specific characteristics of the graphic hardware. Most suited for GPGPU computing are so-called stream algorithms, which perform the same set of operatons on large data sets. Real-time FIR filtering falls into this class of algorithms. It has a high potential for parallelization and a low level of data interdependency. This makes it an ideal candidate for GPGPU computation

RELATED WORK GPGPU

computing has been successfully applied to computationally intensive problems in acoustics and audio processing: This ranges from acoustic simulation methods, like wave-based finite-difference methods (FDMs) [5] as well as geometric acoustics modelling like ray-tracing [6] [7]. It has been applied to sound synthesis [8], spatail sound reproduction systems like wave-field synthesis (WFS) [9] and also music processing [10]. Stingos [11] provides a good overview on applications. Not too many publications on audio signil processing using

GPU can be found in literature. Only a few authors address audio filtering on graphic hardware: Most publiccations deal with straight-forward time-domain implementations of FIR filtering on GPUs. In 2005, Smirnov and Cheah [12] milamented a tapped delay-line (TDL) using fragment shaders on a NVIDIA GeForce 6600 card and compare the performance to an SSE-optimized CPU variant. They conclude that GPU-processing is more efficient for long filters only (>60000 taps). In 2004, Gallo and Stingos [13] introduce techniques for 3D rendering of vieteal scenes using graphic hardware. They use head-related transfer functions (HRTFs) for spatial audio rendering and realize the filtering using simple 4-band equalizers. Their method also allows to realize doppler effects by texture scaling. A more recent timedomain implementation by Kwan and Kripalu's [14] in 2008, also deals with HRTF-based audio rendering of virtual scenes. They convolve audio signals with short HRTFs filters (200 taps) usIng integer-based OpenGL shaders. On a NVIDIA GeForce 8800 GTX card their GPU solution outperforms a CPU convolution sigmilitantly and indicates enough performance to meet real-time condictions. However, they report subtitle problems that come along with integer-processing. In a follow-up paper [15] they remexsurged on a more current NVIDIA GTX 280 card. A recent publiccation by Trevion and Oliveira [16] deals with the implementation of 1D recursive filters on GPUs. The only paper we could find on GPU-based fast convolution in the frequency-domain, is an unpublished course work by Rush [17]. He implements a uniform partitioning on a NVIDIA G80 CPU and considers offline filterIng. Unfortunately, no performance values are presented and the results are not compared to CPU-implementations.

FAST CONVOLUTION ALGORITHM

Fast convolution as a method for efficient FIR filtering has been researched for more than four decades. Several fast convolution algorithms are known today. We found that the choice of algorhythm is even more important when considering GPGPU computeton. Therefore, we first give a brief overview on the methods and discuss their pros and cons. Afterwards we introduce our chosen algorithm.

Brief overview of fast convolution techniques

All fast convolution algorithms have in common, that they callcollate linear convolution efficiently in the frequency-domain, by simple multiplication of discrete Fourier spectra, known as carcollar convolution. The term fast is reasoned by the Fast Fourier Transform (FFT) used to convert between the time- and frequency domain. The original idea was proposed by Stock ham [18] in 1966. His algorithm uses one FFT to convolve two signals (M, N samples). The length of the FFT is chosen so that the convolution result ($M + N - 1$ samples) does not exceed it and time-aliasing is avoided. This algorithm outperforms time-domain filtering (direct-form FIR filters, TDLs) by several magnitudes. However, it has the disadvantage of an input-to-output latency that equals the FFT-length. Moreover, its efficiency drops when long signals are convolved with short ones (many ineffective zeros are processed). These problems can be tackled by choosing a shorter FFT-length and by processing the input data in several steps — either in overlap-add or overlap-save fashion [19]. Still, the FFTlength is at least as long as the filter impulse response and so is the latency. For real-time filtering with long filters, the filter also needs to be partitioned. This allows to freely choose FFT-lengths and thereby to adjust the latency. Two variants are known: In unitfirmly partitioned fast convolution, filters are subdivided into sevaearlsub filters of equal lengths. The overall output is assembled from all sub filter outputs, which need to be delayed accordingly. Kelp [20] demonstrates, how the number of required FFTs/IFFTs can be reduced to one, when delays and sums are implemented directly in the frequency-domain. An DSP-implementation of the algorithm can be found in [21], [22]. The uniformly partitioned fast convolution is also most efficient for offline processing. Here, the FFT-lengths can be optimized in order to minimize the commutational effort. The concept of non-uniformly partitioned fast convolution is relatively new [23]. This algorithm is designed for efficient convolution of long filters (> 1000 filter coefficients) with a short input-to-output delay. Short sub filters are used to minimize the latency, whereas longer sub filters reduce the overall computetonal effort. It can be shown that this algorithm is even signifyscantly more efficient than the uniformly partitioned variant [24]. Details on the implementation can be found in the famous paper by Gardner [25]. But a non-uniform partitioning has also drawbacks: As opposed to a uniform partitioning, the complete filter cannot be exchanged with every processed block.

Basics of audio streaming

In this paper we concern real-time FIR filtering of continuous audie signals. Continuous recording, processing and playback of audie data using computer hardware is performed by audio streamIng. Here, samples are processed and exchanged in units of blocks. All blocks consist of a fixed number of samples, referred to as the streaming block length—or just block length. In this work it is denoted by B. Furthermore, we consider multiple channels. C is the

Length [samples]	Duration [ms]	Data size [bytes]
128	2.9	512
256	5.8	1024
512	11.6	2048

Table 1: Typical properties of sample blocks in real-time audio streaming. Here, a sampling rate of 44.1 kHz and 32-bit floating point samples (4 bytes/sample) are considered.

Convolution algorithm

For audio rendering of complex scenes, non-uniformly partitioned fast convolution is the algorithm of choice. But implementing it on graphic cards can be difficult: The algorithm heavily relies on the ability to process sub filters concurrently. Computation tasks must be priories in order to ensure flawless operation. We circumvent these issues by choosing uniformly partitioned fast convolution for our examinations. It has a constant load balance and does not require asynchronous computations, but is less efficient. The general principle of the algorithm for one channel is illstraded in figure 1. It consists of two main parts: stream processIng and filter processing. Before a filter impulse response can be used for convolution, it has to be transformed into a frequencydomain representation. Therefore, it is uniformly partitioned into filter parts of the block length B. Each filter part is then padded with additional B zeros. We refer to this process as filter packIng. Afterwards, each padded part is FFT-transformed into a disCrete Fourier spectrum. All together, these are then used for the convolution. The stream processing computes the convolution.

GPU IMPLEMENTATION

For each of the algorithm's parts can be parallelized and is thereby a candidate for GPU computation. But we also consider the CPU for computations and will only employ the GPU if it benefits the performance. The question is, where to perform the computations – on the host (CPU) or on the graphics card (GPU). One decision is fixed: The multiplication and addition of DFT spectra the demands the major share of computation and will therefore be computed on the graphics hardware. But we will later see, that for the other parts the choice is not trivial. When implementing the fast convolution algorithm for GPGPU computation, we are faced with some fundamental questons: • How to parallelize the spectrum multiply-adds on the GPU? • Where to perform the FFT/IFFT-transforms? Host or GPU? • How to organize the data structures for maximum efficiency? In the following we present our parallelization approach and address each of the questions in detail. Our parallelization founds on two key principles: • Avoid thread synchronization by avoiding mutual write access on memory locations • Keep data reordering operations to a minimum by using well-arranged data structures

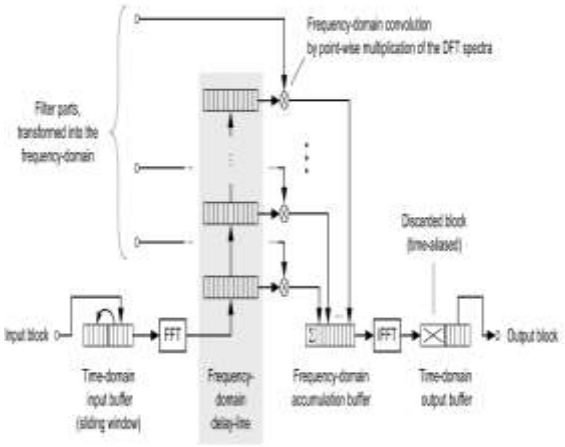

Figure 1: Uniformly-partitioned fast convolution algorithm using the overlap-save scheme. By realizing sub filter delays and the accumulateton of sub filter outputs in the frequency-domain, only one FFT/IFFT-transform is required per processed input block. Of B + 1 complex-valued coefficients each (frequency-domain).

With every stream processing step the data has to be transferred twice — from host-to-device and after calculation from device-tohost. Important to mention here is, that stream processing enforces a synchronous data transfer. The filter data has to be transferred just once (host-to-device), but is usually significantly larger. We consider filters to be partitoned into K parts, each of B filter coefficients. For one channel it consists of K blocks of B filter coefficients in the time-domain or —alternatively—K complex-conjugate symmetric DFT spectra of B + 1 coefficients each. The amount of filter data increases linear with the number of channels.

PERFORMANCE

In this section we analyse the performance of our method for two applications: Time-invariant real-time filtering without the exchange of filters and time-varying filtering. Firstly, we introduce our test system and regard important measures individually — the

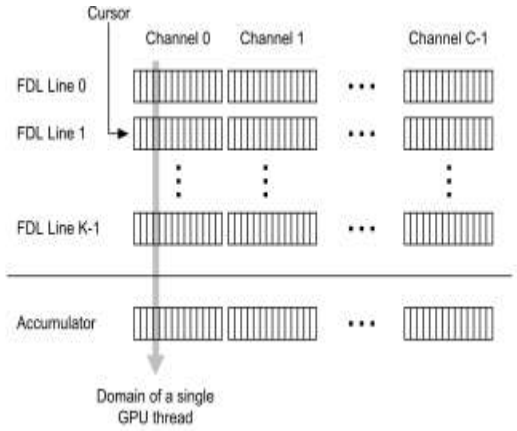

Figure 2: Structure of the frequency-domain delay-lines (FDLs) and the frequency-domain accumulation buffers. The greyed-out vertical bar illustrates the range of DFT coefficients, a single GPU thread is responsible for.

data transfer and computation of Fast Fourier Transforms. Afterwards, we present the performance measures for static filtering. Dynamic filtering cannot be expressed by a single performance value, because it depends on opposing parameters: The number of channels and filter length versus the desired filter update rates. Here we consider the performance by means of an example csenaira and discuss how these parameters relate.

Test system

The test system used is a dual quad-core machine, with two intell Xeon X5570 (Gaines town) processors [27] running at 2.93 GHz. Each processor has 8 MB of shared L3-Cache available. The machine has 4 GB of DDR3-1333 memory. The graphic card is an NVIDIA GeForce GTX 285 [28]. It features the NVIDIA GT200b graphic chip, which has 240 SPs in total, each clocked at 1476 MHz They are arranged in 30 SMs. Our card has 1024 MB GDDR3 video memory, clocked at 1242 MHz and linked via 512-bit memory interface. It uses a PCI-Express 2.0 BUS interface with 16 lanes (x16), resulting in a theoretical bus bandwidth of 8 GB/s. The operating system is Microsoft Windows XP Profissional (32-bit). We built our software using the Microsoft Viseal Studio 2005 (SP1) C++ compiler. We use Streaming SIMD Extensions (SSE) along with a 16-byte structure alignment. The CUDA version used is 2.3. For FFTs on the host we employ the FFTW library [29] version 3.2.2. All tests were carried out on 32- bit single precision floating points. High-precision timing was reallied using the Win32-function QueryPerformanceCounter. We used an RME Hammer fall audio device and Sheinberg's ASIO interrace [30] for low-latency audio streaming.

Data transfer

In advance of any calculation on the GPU, the required data must first be copied onto the graphic card and after the calculation is finished the results must be read back to the host. Data transfer is fundamental — especially

for real-time processing. Every micosecant that is spent on data transfer, cannot be used for GPU calculations. Asynchronous data transfers can be used to mask

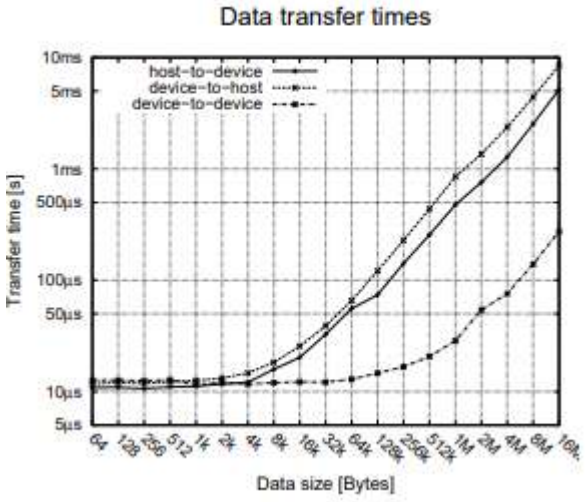

Figure 3: Data transfer times measured on the test system.

CONCLUSIONS

We have presented an implementation real-time FIR on graphices hardware. Our method achieves an outstanding performance, which exceeds all prior implementations. On a NVIDIA GTX 285 card we successfully realized static filtering of more than 200 channels with individual filters of more than 40.000 coefficients at an input-to-output latency of less than 6 ms. We like to point out, that all measures have been benchmarked under stable operaton. The theoretical peak performance is even higher, because this performance was achieved by just utilizing 44% of the computeton power of our GPU. This is reasoned by subtitle problems that emerge under real-time conditions. For a stable operation, variatons in the processing runtimes must be tolerated. In order to prevent dropouts, not the full-time budget time can be exhausted for computations. We discovered randomly occurring system latencies of approximately 1 ms. These turned out to be critical especially for very low latency applications. We also analysed time-varying filtering in detail, where filters are exchanged during runtime. Dynamic filtering demands more computation and relies even more afast data transfer. Neverthe less, for 64 channels and 1,0 s filters, the full filter set (all 64 channels at once) can still be updated with over 40 Hz. This is impressive, but we like to state the PCI-Express BUS can still be a bottleneck. When many long filters are exchanged synchronously, additional filter exchange latencies occur. However, this has minor relevance for applications in practice. Here, typical filter update rates are in the range of 50-100 Hz. Moreover, long filters are usually not entirely exchanged with such high rates [1].

REFERENCES

[1] Tobias Lentz, Dirk Schroder, Michael Verlander, and Ingo Casemaker, "Virtual reality system with integrated sound field simulation and reproduction," EURASIP Journil on Advances in Signal Processing, vol. 2007, available at http://downloads.hindawi.com/journals/asp/2007/070540.pdf.

[2] J.D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kruger, A.E. Lefohn, and T.J. Purcell, "A survey of general-purpose computation on graphics hardware," in proceedings of EUROGRAPHICS, 2005, available at http://www.idav.ucdavis.edu/func/return_pdf?pub_id=907.

[3] "NVIDIA CUDA Programming Guide Version 2.3," available at http://developer.download.nvidia.com/compute/cuda/2_3/toolkit/docs/NVIDIA_CUDA_Programming_Guide_2.3.pdf.

[4] "ATI Close-To-Metal (CTM) Guide Version 1.01," available at http://ati.amd.com/companyinfo/researcher/documents/ATI_CTM_Guide.pdf.

[5] Nikunj Raghuvanshi, Nico Galoppo, and Ming C. Lin, "Accelerated wave-based acoustics simulation," in SPM '08: Proceedings of the 2008 ACM symposium on Solid and physical modeling, New York, USA, 2008, pp. 91–102, ACM.

[6] Marcin Jedrzejewski and Krzysztof Marasek, "Computation of room acoustics using programmable video hardware," in International Conference on Computer Vision and Graphics ICCVG'2004, Warsaw, Poland, 2004.

[7] Niklas Röber, Ulrich Kaminski, and Maic Masuch, "Ray acoustics using computer graphics technology," in Conference on Digital Audio Effects (DAFx-07) proceedings, Bordeaux, France, 2007.

[8] Qiong Zhang, Lu Ye, and Zhigeng Pan, "Physically-based sound synthesis on GPUs," Entertainment Computing-ICEC 2005, pp. 328–333.

[9] Dimitris Theodoropoulos, Catalin Bogdan Ciobanu, and Georgi Kuzmanov, "Wave field synthesis for 3d audio: architectural prospectives," in CF '09: Proceedings of the 6th ACM conference on Computing frontiers, Ischia, Italy, 2009.

[10] Eric Battenberg and David Wessel, "Accelerating NonNegative Matrix Factorization for Audio Source Separation on Multi-Core and Many-Core Architectures," in 10th International Society for Music Information Retrieval Conference (ISMIR 2009), Kobe, Japan, 2009.

[11] Nicolas Tsingos, "Using programmable graphics hardware for acoustics and audio rendering," in proceedings of the 127th AES Convention, New York, USA, 2009.

[12] Alexey Smirnov and Tzi-cker Chiueh, "An Implementation of a FIR Filter on a GPU," 2005, available at http://www.ecsl.cs.sunysb.edu/fir/fir.ps.

[13] Emmanuel Gallo and Nicolas Tsingos, "Efficient 3D audio processing with the GPU," in GP2, ACM Workshop on General Purpose Computing on Graphics Processors, 2004.

[14] Brent Cowan and Bill Kapralos, "Spatial sound for video games and virtual environments utilizing real-time gpubased convolution," in Proceedings of the 2008 Conference on Future Play, New York, NY, USA, 2009, ACM.

[15] Brent Cowan and Bill Kapralos, "Real-time gpu-based convolution: a follow-up," in Proceedings of the 2009 Conference on Future Play on @ GDC Canada, Vancouver, Canada, 2009.

[16] F. Trebien and M.M. Oliveira, "Realistic real-time sound resynthesis and processing for interactive virtual worlds," The Visual Computer, vol. 25, no. 5, pp. 469–477, 2009.

[17] Michael Rush, "Convolution engine utilizing NVIDIA's G80 processor," available at http://www.ece.ucdavis.edu/mmrush/mmrush_eec277_final ˜ _writeup.pdf.

[18] T.G. Stockham Jr, "High-speed convolution and correlation," in Proceedings of the April 26-28, 1966, Spring joint computer conference. ACM, 1966.

[19] Alan V. Oppenheim and Ronald W. Schafer, Discrete-Time Signal Processing, Prentice Hall Signal Processing Series. Prentice Hall, 1989.

[20] Barry D. Kulp, "Digital equalization using fourier transform techniques," Journal of the Audio Engineering Society, 1988.

[21] Anders Torger and Angelo Farina, "Real-time partitioned convolution for ambiophonics surround sound," IEEE Workshop on the Applications of Signal Processing to Audio and Acoustics, pp. 195–198, 2001.

[22] E. Armelloni, C. Giottoli, and A. Farina, "Implementation of real-time partitioned convolution on a DSP board," pp. 71–74, 2003.