FAST AND EFFICIENT REAL-TIME GPU BASED IMPLEMENTATION OF WAVE FIELD SYNTHESIS

Rishabh Ranjan and Woon-Seng Gan

School of Electrical and Electronics Engineering, Nanyang Technological University, Singapore
{rishabh001, ewsgan}@ntu.edu.sg

ABSTRACT

Wave Field Synthesis (WFS) aims to replicate true sound field in an extended listening area with the help of loudspeaker arrays. WFS practical setups are heavily computational, as they need to drive many loudspeakers to accurately render multiple virtual sources. Thus, performance bottleneck occurs due to the sequential implementation on PCs with few cores. In addition, real-time spatial audio reproduction systems like WFS are subjected to hard real-time constraints, limiting system throughput and require cascading of several PCs to improve performance. In this paper, a fast and efficient graphics processing unit (GPU) based implementation of WFS is proposed to enhance the system throughput by extracting maximum data parallelism in the algorithm. The proposed method, implemented on NVidia C2075 GPU, uses block based partitioning approach to achieve peak system throughput of 1,400 Msamples per second, while rendering up to 200 real-time sound sources.

Index Terms— WFS, GPU, Parallel processing

1. INTRODUCTION

Wave Field Synthesis (WFS) is a spatial audio reproduction technique capable of reproducing high fidelity sound in large listening area with the help of loudspeaker arrays [1]. Listeners get to experience realistic sound scene as they are free to move in the listening area and virtual sources are localized as close as possible to their true positions. In practice, such high fidelity systems require many driving units, while rendering multiple virtual sources, making WFS a heavy computationally complex system. Commercially viable solutions from SonicEmotion [2] and IOSONO [3] can render up to 64 real-time sources for 24 and 32 driving units, respectively. Furthermore, before hardware implementation can be realized, the synthesized sound field quality needs to be analyzed across the entire listening area. Thus, two processing blocks, namely, synthesized signal block and sound field synthesis block are added to the system, which can be used for real-time analysis of a WFS set up. Collectively, we call such system: a three-fold WFS set up. Overall, WFS is a highly parallel data intensive application but suffers from limited resource problem and low throughputs when implemented on today's multi-core PC platforms.

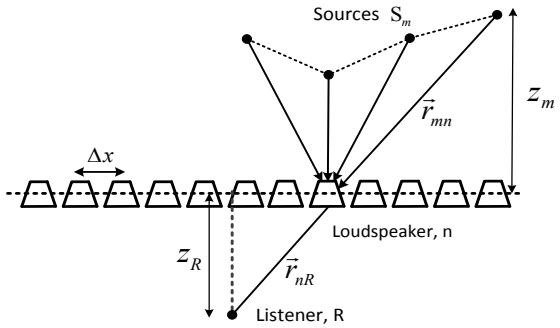

Figure 1 Geometry for WFS equations

With the advent of graphics processing units (GPU), maximum resource utilization can be achieved using parallel computing architecture. Recently, modern GPUs like GTX590, C2075, K10 etc. have hundreds to thousands of processing cores, which can handle massively parallel and computationally intensive applications such as WFS. Essentially, algorithms written for small-scale multicore PCs need to be sufficiently parallelized and adapted for multi-threading architecture to take full advantage of today's GPUs. Additionally, in real-time spatial audio applications, GPU must process the audio data within a fixed time interval, while also taking account of the data transfer overheads. This makes parallelization the most critical task for performance.

In this paper, we present a generic real-time implementation of three-fold WFS set up on GPU using CUDA [4] technology with MATLAB [5]. Low level parallel programming language, CUDA is used to achieve peak performance by giving complete control of GPU architecture to the user. Concretely, the main objective of this work is to develop a fast real-time implementation of WFS set up by efficiently mapping the massive data parallelism into WFS and thereby, taking advantage of running thousands of threads in parallel. Results show that peak system throughput of 1,400 Msamples per second (MSPS) can be achieved with 20 folds improvement over CPU based implementation.

This paper is organized as follows. Related work is mentioned in Section 2. Section 3 briefly outlines the overview of WFS governing equations followed by block wise processing of real-time WFS set up in Section 4. Section 5 further examines the optimization techniques and overhead reduction methods in GPU. Section 6 shows the experimental results with key findings reported in the concluding Section 7.

978-1-4799-2893-4/14/$31.00 ©2014 IEEE

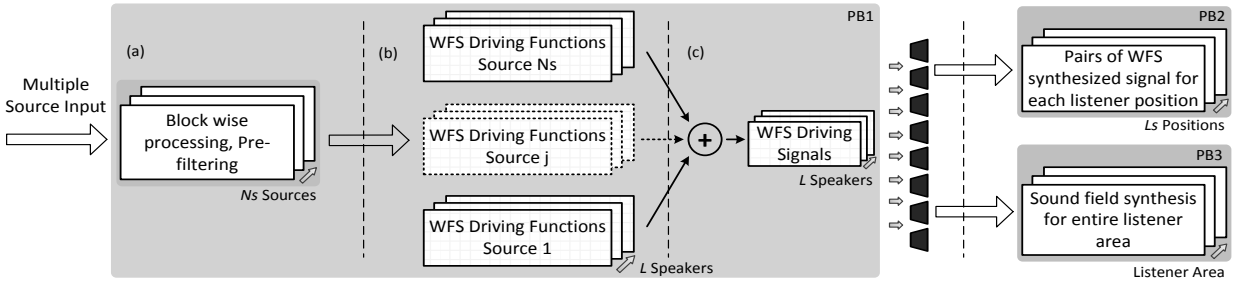

Figure 2 Real-time WFS processing with processing blocks (PB1, PB2 and PB3)

2. RELATED WORK

Due to the advent of more powerful GPU, we are seeing new works related to the real-time spatial audio processing applications like GPU on WFS platforms. Theodoropoulos et al. [6, 7] implemented WFS on different multicore platforms including GPU with the focus on architectural perspectives of these platforms. They reported speed up of around 10-20 times on GPU against Intel core 2 duo PC and estimated up to 64 real-time sources rendering for 96 loudspeakers. In [8], real time implementation of WFS was proposed on GPU and CUDA using NU-Tech framework [9] with peak speed up achieved up to around 4 times, although there is no mention of number of real-time source rendering. In [10], authors implemented WFS and a room compensation block with added computational complexity on three different GPU platforms. Their implementation achieved real-time rendering up to 50, 80 and 300 sources for Tesla, Fermi and Kepler architecture, respectively when room compensation was not applied for 96 loudspeakers. In contrast to above works, our implementation is based on hybrid time-frequency approach, which has lesser computational complexity. In addition, two processing blocks are also implemented in GPU for sound field analysis. System throughput is used as a better measure of system performance instead of speed up. In the end to end comparison with [10] and our Fermi GPU, we obtained around 4 fold improvement in number of real-time sources for 96 driving units.

3. WFS OVERVIEW

WFS is a multichannel spatial audio reproduction system, which works on the principle of natural wave propagation as derived from Huygens principle by Berkhout [11, 12]. WFS driving signals (loudspeaker signals) are derived from discrete 2D Rayleigh integral using stationary phase approximation [12-14] as

$$D(n,w) = \sum_m S_m(w) \sqrt{\frac{jw}{2\pi}} \sqrt{\frac{Z_{ref}}{Z_m + Z_{ref}}} \frac{Z_m}{|\overrightarrow{r_{mn}}|} \frac{e^{-j\frac{w}{c}|\overrightarrow{r_{mn}}|}}{\sqrt{|\overrightarrow{r_{mn}}|}}. \quad (1)$$

Each driving signal is the total contribution from delayed and weighted samples of all the pre-filtered source signals as shown in Figure 1. Z_{ref} is usually taken as reference listener distance in the center of the listener area for the calculation of driving signals. Synthesized sound pressure at any listener point, *R* and any time, *t* is given using driving signals [12, 13] as shown in (2)

$$p(R,t) = -\frac{1}{4\pi} \sum_n \frac{1}{|\overrightarrow{r_{nR}}|} d\left(n, t - \frac{|\overrightarrow{r_{nR}}|}{c}\right) \Delta x, \quad (2)$$

where d (•) is the inverse Fourier transform of the driving signal in (1). Clearly, synthesized sound pressure is given by contributions from all the driving signals summed up at the listener position.

4. REAL-TIME IMPLEMENTATION

The three-fold WFS spatial reproduction set up is shown in Figure 2. Using a linear array of loudspeakers, driving signal (block PB1) for each loudspeaker is governed by (1) and subsequently, used in synthesis function (2) altogether for processing blocks PB2 and PB3, as shown in Figure 2.

As stated earlier, the processing blocks PB2 and PB3 can be used to assess synthesized sound field quality for different kinds of WFS set up. PB2 computes the virtually synthesized binaural signals at *Ls* listener positions for real-time playback over headphones. PB3 synthesizes snapshots of the sound field in the entire listening area ($dimx \times dimz$) for different test signals. $dimx \times dimz$ represents the number of sample points in the entire listener area. These snapshots can also be used for analysis of several artifacts, like spatial aliasing, truncation effects and amplitude errors.

For real-time processing, one frame of audio data must be processed within the t_{frame} (frame size/sampling frequency). At the same time, we should aim to maximize the system throughput by processing more data within t_{frame}. The real time implementation of WFS set up is based on overlap-save technique with 50 % overlap using frame size of *M* samples and *M* previous samples. As shown in Figure 2, driving signal block (PB1) is divided into three stages, namely, (a) pre-filtering for multiple sources, (b) individual driving signals due to all sources at each loudspeaker, and, (c) compute driving signals using reduction sum of output matrix at stage (b). Real-time filtering using block convolution is generally faster in frequency domain [10]. Therefore, pre-filtering is implemented in frequency domain, while the rest of the stages have been implemented in spatio-temporal domain. Recent contributions [15, 16] have also shown that real-time filtering of multiple data can be processed concurrently on GPU. Pre-filtering is carried out by element-wise complex multiplications of 2*M*-FFT transformed multiple source data. Table 1 summarizes the computational complexity of different computations stages in PB1. Clearly, time-frequency approach seems to have the lowest complexity for larger values of virtual sources (*Ns*) and speakers (*L*). Both

Table 1 Computational Complexity of different computation stages in PB1 (MAD: Multiply/Addition; ADD: Addition)

Stage	time-frequency	time [6,7]	frequency [8,10]
FFT (a)	$2M \times Ns$	-	$2M \times Ns$
MAD (a)	$12M \times Ns$	$8M^2 \times Ns$	$12M \times Ns$
IFFT (a)	$2M \times Ns$	-	-
MAD (b)	$L \times M \times Ns$	$L \times M \times Ns$	$8L \times 2M \times Ns$
ADD (c)	$L \times M \times Ns$	$L \times M \times Ns$	$2L \times 2M \times Ns$
IFFT	-	-	$2M \times L$

frequency and time domain approaches have high arithmetic density due to complex arithmetic operations and circular convolutions, respectively, resulting in higher complexity. Blocks PB2 and PB3 are also implemented in time-domain using weighted and delayed contribution from driving signals at listener positions with computational complexity $L \times M \times 2Ls$ and $L \times dimx \times dimz$ respectively.

5. GPU IMPLEMENTATION

Most of the audio processing is done in GPU using low level CUDA programming language along with MATLAB as host environment, controlling the GPU execution. Recently, MATLAB has added the support for GPU computing to its parallel computing toolbox (PCT) to take advantage of the parallel computing from MATLAB environment [17]. MATLAB along with the CUDA kernels [18] serves as a useful tool for the fast development of existing MATLAB applications onto GPU using custom CUDA functions, as well as overloaded MATLAB functions for GPU.

The datasets to be computed are carefully partitioned into multiple contiguous blocks to take advantage of the data reuse using the on-chip cache and exploit coalesced memory access as much as possible. WFS algorithm is also segregated into parallel functions to exploit maximum data parallelism. Other CPU-GPU optimizations include shared memory, constant memory, data reorganizations, and overlapped executions are also taken into account to further speed up the processing time. We will now describe the implementation of each parallel task on GPU along with the optimization choices made for the best performance.

5.1. Pre-filtering of multiple sources

Computational complexity of this task is in $O(2M \times Ns)$, where *Ns* is the number of sources. A total of $2M \times Ns$ threads are launched with thread block size of 256 threads. Each thread computes one complex multiplication for a single source sample with the corresponding filter coefficient. Shared memory is used to synchronize the common filter coefficients within a thread block. Both MATLAB built-in overloaded FFT function for GPU, as well as NVIDIA CUFFT library [4] are considered, since both can perform frequency transformations for multiple sources concurrently.

5.2. Driving signals computation

Driving signals are computed in time domain after taking inverse Fourier transform and discarding first invalid *M* samples from output at stage (a). The current *M* pre-filtered samples are then merged with previous 2,048 samples to form pre-filtered source buffers (to access delayed samples of driving signals). As mentioned in Section 4, driving signals computation is further divided into two stages to extract the maximum data parallelism. First, individual driving signals are computed as three-dimensional output matrix of size $(L \times M \times Ns)$. Kernel is launched with $L \times M \times Ns$ threads with two dimensional thread blocks of size $l \times m$ threads, corresponding to m samples of l driving signals. A thread block computes these $l \times m$ samples with each thread computing one sample. Weight and delay values are computed once for each speaker position and are reused using shared memory within a thread block. Pre-filtered source samples are also transferred to shared memory and shared across a thread block to further reduce the memory latency.

Table 2 Average execution times for WFS processing blocks (*Ns* =1, *L* = 161, *M* = 512, *Ls* = 1, *dimx* = *dimz* = 256)

Platform	PB1 (msec)	PB2 (msec)	PB3 (msec)
CPU	1.94	2.08	745.5
CPU+GPU	1.56	0.37	2.7

Finally, driving signals for each loudspeaker are computed as separate CUDA kernel using reduction sum across the third dimension of the output matrix at stage (b). Since reduction sum is a sequential operation, kernel with $L \times M$ threads will result in very low throughput with each thread performing Ns serial additions. We parallelize the reduction sum using binary tree based parallel reduction [19], where partial sums are computed in parallel and synchronized within thread block. Kernel is thus, launched with one dimensional thread block of size *Ns* and grid size of $L \times M$ thread blocks. Each thread block computes one sample of a driving signal and result is written back to global memory.

5.3. Synthesized signals and Sound field synthesis computation

Similar to the driving signals computation, processing blocks PB2 and PB3 are implemented in GPU by launching two separate kernels, one for computations of weighted and delayed driving signals and other for the parallel reduction sum (see Section 5.2). First kernel is launched with $L \times M \times 2Ls$ threads and $L \times dimx \times dimz$ threads respectively, for PB2 and PB3. For second kernel, each thread block (of size *L* threads) computes one sample of synthesized signal at a given listener position using parallel reduction sum for both the processing blocks.

Other optimizations include constant memory to store speaker and source positions, overlapped executions on host side for data transfer as well as data rearrangements.

6. RESULTS

Our processing platform consists of the Intel quad core i7 processor as CPU, and Fermi architecture based C2075 as 448-core GPU with 14 simultaneous multiprocessors (SMs). We analyze the performance of the real-time WFS set up based on implementation aspects, like latency and throughput of the system as well as algorithmic complexity. It should be noted that the CPU implementation inherently takes

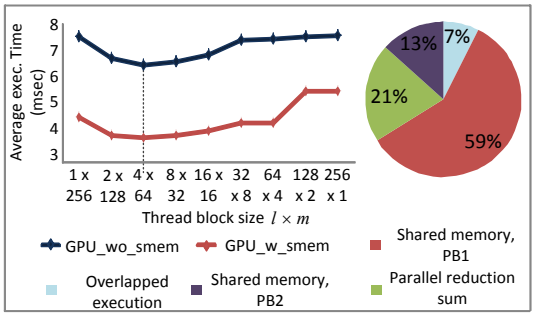

Figure 3 Impact of major optimization techniques over GPU un-optimized implementation (Ns =100, L = 161, M = 512, Ls = 1)

advantage of the multicore host architecture and multithreading by MATLAB inbuilt functions.

Number of speakers and sound sources are the two main parameters, which control the real-time performance of the WFS driving function block both in terms of efficiency and behavior. In order to create a realistic and practiced WFS system, multiple sources rendering over huge loudspeaker array is required. But, a real-time implementation poses constraints on the number of loudspeakers and sources, and often there is a tradeoff between performance and behavior of the system. Fewer loudspeakers can result in spatial aliasing, while limited number of virtual sources may not give an enriching sound experience to the listeners. Since modern GPUs are capable of running thousands of threads in parallel by exploiting massive data parallelism inherent in an application, real-time performance can be improved significantly, while at the same time achieving desired sound field quality.

Table 2 shows the average execution time per frame for the three processing blocks rendering a single source. Execution times reported for GPU is inclusive of the data transfer between host and device. Upon dividing the reported execution time by the number of samples processed for each block, PB3 is clearly identified as the slowest block. It executes 275 times fast after GPU optimization, which is mainly due to the inherent massive data parallelism involved in the computation of synthesized signals. On the other hand, PB1 is the slowest block after GPU optimization given the lack of much parallelism in single source rendering. For driving function block, the number of samples processed is $L \times M \times Ns$ while, for the other two blocks, it is same as their complexity. GPU efficiency can be considerably improved when there are many sources to be rendered by extracting more data parallelism. However, increasing the workload on GPU will also incur high global memory overhead. As discussed in Section 5, several optimization techniques can be applied to speed up the system. Figure 3 shows the impact of major optimizations on system performance. As shown, after shared memory optimization, kernel for stage (b) executes twice as fast as one without any optimization. It is also shown that how the different choices of thread block configuration affect the execution time. As shown, there is a trade-off between choices of l and m, with optimum thread block configuration found to be 4×64 for a fixed block size of 256 threads. This is mainly due to the extra memory overheads or

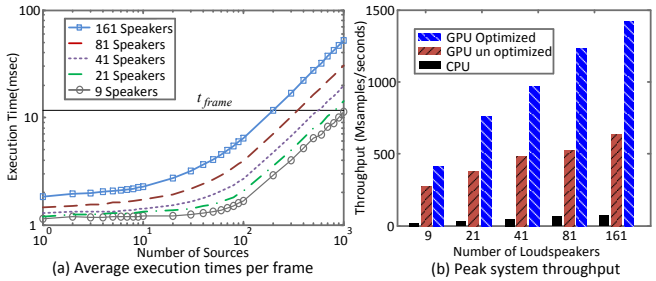

Figure 4 Average Execution times and Peak throughput of overall system (PB1 + PB2)

more arithmetic operations at the two extremes choices of l and m. Similarly, optimum thread block sizes for other kernels have been found. From the pie chart shown in Figure 3, shared memory with optimum thread block size has most of the impact in improving the GPU performance with 59% share for the driving function block PB1. However, for block PB2, there is only 13% improvement over non-optimized GPU implementation. This is mainly due to the fewer data parallelism present in PB2 as compared to PB1. Another significant effect is due to the parallel reduction sum especially, if CUDA kernel is launched with thousands of threads. Finally, overlapped execution, which was used to perform some of the data transfers and host processes simultaneously with kernel execution, also resulted in 6% latency savings.

Figure 4 shows the average execution times per frame and peak throughput of the overall system for blocks PB1 and PB2. As shown, processing time must be less than t_{frame} for WFS set up to perform in real-time (Figure 4). Thus, GPU can render up to 1,000 real-time sources for 9 speakers or 200 real-time sources for 161 speakers. At the same time one can also listen to the synthesized signals in real-time. System throughput is calculated as number of sampled processed per unit time. As shown in Figure 4, optimized GPU implementation can have peak system throughput of 1,400 MSPS almost twice of the un-optimized GPU implementations and 20 times of the CPU based implementation. It is also important to note that throughput can be substantially improved by increasing the GPU workload and hence, exploiting the maximum data parallelism in GPU.

7. CONCLUSION

In this paper, we presented a fast and efficient GPU implementation for a real-time wave field synthesis system. We have successfully accelerated the overall WFS set up with peak throughput of 1,400 MSPS using several GPU optimization techniques. We achieved 20-fold improvement over CPU based implementation, while up to 200 sources can be rendered in real-time with 161 loudspeaker array. In addition, for high-end GPUs running thousands of parallel threads, we are able to synthesize WFS signals at any listener positions in real-time with as many sources. One of the main features of this work is that all the audio processing is done in GPU, while CPU is freed for other independent tasks like IO buffering or overlapped data transfers. Among the several optimization techniques, shared memory provides us the most significant performance improvement.

8. REFERENCES

[1] D. de Vries, "Wave Field Synthesis," in *AES Monograph*, New York, 2009.

[2] SonicEmotion. [Online]. Available: http://www.sonicemotion.com/home

[3] IOSONO. [Online]. Available: http://www.iosono-sound.com/

[4] NVIDIA, "NVIDIA CUDA C Programming Guide— v4.2," April 2012.

[5] T. Mathworks. MATLAB: The Language of Technical Computing. [Online]. Available: http://www.mathworks.com/products/matlab/index.html

[6] D. Theodoropoulos, C. B. Ciobanu, and G. Kuzmanov, "Wave field synthesis for 3D audio: architectural prospectives," in *ACM International Conference on Computing Frontiers*, 2009, pp. 127-136.

[7] D. Theodoropoulos, G. Kuzmanov, and G. Gaydadjiev, "Multi-Core Platforms for Beamforming and Wave Field Synthesis," *IEEE Transactions on Multimedia,* vol. 13, pp. 235-245, 2011.

[8] A. Lattanzi, E. Ciavattini, S. Cecchi, L. Romoli, and F. Ferrandi, "Real-Time Implementation of Wave Field Synthesis on NU-Tech Framework Using CUDA Technology," in *128th AES Convention*, London, 2010.

[9] A. Lattanzi, F. Bettarelli, and S. Cecchi, "NU-Tech: the entry tool of the hArtes toolchain for algorithms design," in *124th AES Convention,* , Amsterdam, The Netherlands, 2008, pp. 1-8.

[10] J. A. Belloch, M. Ferrer, A. Gonzalez, J. Lorente, and A. M. Vidal, "GPU-Based WFS Systems with Mobile Virtual Sound Sources and Room Compensation," in *Audio Engineering Society Conference: 52nd International Conference: Sound Field Control-Engineering and Perception*, 2013.

[11] A. J. Berkhout, "A holographic approach to acoustic control," *Journal of the Audio Engineering Society,* vol. 36, pp. 977-995, 1988.

[12] A. J. Berkhout, D. de Vries, and P. Vogel, "Acoustic control by wave field synthesis," *The Journal of the Acoustical Society of America,* vol. 93, p. 2764, 1993.

[13] R. Ranjan and W. S. Gan, "Wave Field Synthesis: The Future of Spatial Audio," *IEEE Potentials,* vol. 32, pp. 17-23, April 2013.

[14] P. Vogel, "Application of wave field synthesis in room acoustics," PhD Thesis, Delft University of Technology, 1993.

[15] 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, pp. 449-457, 2011.

[16] J. A. Belloch, M. Ferrer, A. Gonzalez, F.-J. Martínez-Zaldívar, and A. M. Vidal, "Headphone-based Spatial Sound with a GPU Accelerator," *Procedia Computer Science,* vol. 9, pp. 116-125, 2012.

[17] T. Mathworks. *MATLAB GPU Computing Support for NVIDIA CUDA-Enabled GPUs*. Available: http://www.mathworks.com/discovery/matlab-gpu.html

[18] MATLAB-CUDA. CUDA kernel integration in MATLAB applications [Online]. Available: http://www.mathworks.com/help/distcomp/executing-cuda-or-ptx-code-on-the-gpu.html

[19] M. Harris, S. Sengupta, and J. D. Owens, "Parallel prefix sum (scan) with CUDA," *GPU gems,* vol. 3, pp. 851-876, 2007.