There and Back Again: The Practicality of GPU Accelerated Digital Audio

Harri Renney
Com Sci Research Centre
University of West of England
Bristol, UK
harri.renney@uwe.ac.uk

Benedict R. Gaster
Com Sci Research Centre
University of West of England
Bristol, UK
benedict.gaster@uwe.ac.uk

Thomas J. Mitchell
Creative Technology Lab
University of West of England
Bristol, UK
tom.mitchell@uwe.ac.uk

ABSTRACT

General-Purpose GPU computing is becoming an increasingly viable option for acceleration, including in the audio domain. Although it can improve performance, the intrinsic nature of a device like the GPU involves data transfers and execution commands which requires time to complete. Therefore, there is an understandable caution concerning the overhead involved with using the GPU for audio computation. This paper aims to clarify the limitations by presenting a performance benchmarking suite. The benchmarks utilize OpenCL and CUDA across various tests to highlight the considerations and limitations of processing audio in the GPU environment. The benchmarking suite has been used to gather a collection of results across various hardware. Salient results have been reviewed in order to highlight the benefits and limitations of the GPU for digital audio. The results in this work show that the minimal GPU overhead fits into the real-time audio requirements provided the buffer size is selected carefully. The baseline overhead is shown to be roughly $0.1ms$, depending on the GPU. This means buffer sizes 8 and above are completed within the allocated time frame. Results from more demanding tests, involving physical modelling synthesis, demonstrated a balance was needed between meeting the sample rate and keeping within limits for latency and jitter. Buffer sizes from 1 to 16 failed to sustain the sample rate whilst buffer sizes 512 to 32768 exceeded either latency or jitter limits. Buffer sizes in between these ranges, such as 256, satisfied the sample rate, latency and jitter requirements chosen for this paper.

Author Keywords

NIME, DMI, GPGPU, HPC

CCS Concepts

• **Computing methodologies → Graphics processors;**

1. INTRODUCTION

General-purpose GPU (GPGPU) computing provides the capacity for massively parallel processing using a widely available hardware accelerator: the graphics processing unit (GPU). There are many digital audio processes that are suitable for the GPGPU environment and can result in a substantial performance increase, relative to a CPU bound program. Examples of academic work exploring the use of GPUs in digital audio with notable results can be seen in [17] and [1]. However, the communication overhead imposed when using a hardware accelerator, like the GPU, can outweigh the benefits. This is especially relevant for applications with real-time requirements, such as digital audio processing. There seems to be an understandable caution within the digital audio developer community surrounding GPGPU, and consequently GPGPU optimisation is sparsely used. But what are the practical limitations of audio processing on the GPU? This paper aims to investigate the performance overhead of GPGPU within the audio domain, by measuring the communication overhead between the CPU and GPU in both unidirectional and bidirectional cases. In particular, the contributions are:

- An open-source benchmarking suite for evaluating GPU computation within a digital audio domain;
- Results from the benchmarking suite across various hardware systems; and
- A summary of the salient findings.

1.1 Digital Audio

Digital audio is the representation of acoustic sound in a discretized and quantized form in order for it to be computable [2]. An originally continuous audio signal must be broken up into a finite set of samples representing the signal (quantization), where each sample is represented with finite precision (discretization). When signals are represented in this form, computer systems can process existing or synthesise new signals.

The most convenient, and abundant way to process digital audio is to program tools and software that runs on the central processing unit (CPU) in a language such as C++. The CPU has limitations as a powerful, but coarse-grained parallel processor, as shown when compared with other processors by Mistry et al. [9]. Highly regarded experts predict future computational growth will come from utilizing parallel architectures and heterogeneous computing [16]. As a result, processor design is undergoing significant changes. As the state of the art hardware develops, software must map appropriately. There is a variety of fundamentally different processor types that can be used in digital audio processing. These include digital signal processors (DSP), field-programmable gate arrays (FPGA) and graphics processing units (GPU). In this work, we explore the practicalities of using the GPU as a device for offloading suitable tasks in digital audio.

1.2 General-Purpose GPU Computing

GPGPU computing is the use of the GPU, originally intended for rendering graphics, for general computation [8]. For a long time, GPU architectures combined with the available software APIs, required general compute problems to

NIME'20, July 21-25, 2020, Royal Birmingham Conservatoire, Birmingham City University, Birmingham, UK.

Requirement	Recommended	Limit
Sample Rate	96000	44100
Latency	10ms	20ms
Jitter	±1ms	±3ms

Table 1: Real-time audio requirements.

be mapped into the graphics domain. This non-trivial mapping was often not practical and was mostly only pursued by academics. Over time, GPGPU software standards and APIs were proposed and have been developed with great success. Owens et al. [12] cover the development of GPGPU from its origins where Mark Harris first coined the term GPGPU, to the mature development environments available today that are under continuous development in both industry and academia.

1.3 Real-time Processing

Real-time processing is the requirement for a program to meet a set of performance requirements consistently for a particular application. Requirements vary between applications, where variable amounts of data need to be processed within a fixed and inflexible time frame. In the case of real-time audio, a consistent number of audio samples needs to be produced every second. The real-time requirement in audio is very strict, as even a few missed samples or delays results in instantly noticeable 'glitches'. Latency is a term used in various fields to describe the delay between the initiation of an event to its conclusion. Within the context of this work, we focus on the time taken for a buffer of audio samples to be dispatched and returned from the CPU to the GPU. Although this avoids necessary stages in digital audio, like the operating system and sound devices, restricting this measurement isolates the GPU overhead specifically. If the latency is too high in a real-time, interactive application, it will often detriment performance. The other impacting factor for real-time audio is known as jitter. In this work, jitter will refer to the variation in latency between consecutive buffers dispatched to the GPU.

The standard requirements surrounding real-time audio has long been debated. In this paper, the real-time requirements discussed in [7] and [6] are used. These are shown in Table 1.

2. TECHNIQUES

In this section, the general techniques used in the design of GPGPU applications are briefly covered. These are important concepts or optimizations which help to maximize the benefits of using the GPU.

2.1 Buffering

Buffering is an important technique used to reduce the communication overhead considerably between the CPU and GPU. It works by requesting the GPU to execute and generate a variable sized buffer of audio samples each time, rather than a single sample. Figure 1 visualizes buffer transfers to and from the GPU for computation.

The ideal buffer size for GPU dispatch is an extremely important factor in the performance within real-time applications. This paper aims to explore the limits for the range of buffer sizes that work within the constraints of real-time audio.

2.2 Unified Memory

Discrete GPUs are independent devices typically included in a system as an extra peripheral. Therefore, discrete GPUs

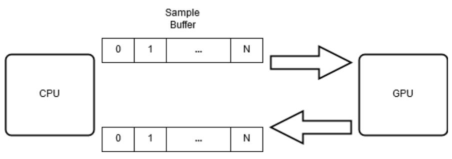

Figure 1: Buffers of N samples are dispatched to GPU and returned to CPU.

have their own local memory and communicate with the CPU over system buses, shown in Figure 2. System buses take time to transfer data and introduce unavoidable latency, irrespective of the data size.

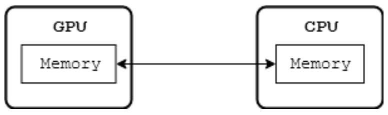

Figure 2: CPU to GPU memory accesses across system bus into respective device's memory.

Unified memory is a single addressable memory space accessible by separate processors, shown in Figure 3. This means the overhead of transferring over the system bus is avoided. Integrated GPUs take advantage of this technology and share a memory space with the CPU. This allows fast memory transfers between CPU and integrated GPU by default, which is an important perspective to consider for this paper.

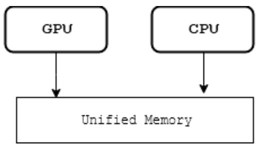

Figure 3: CPU and GPU memory accesses using shared, unified memory.

2.3 Pinned Memory

Pinned memory, or page-locked memory is a special type of memory that is located within the CPU host memory and GPU memory at the same time. Changes in the host memory can take effect in the GPU memory which can be accessed by the GPU for computation. In essence, pinning memory removes any extra step in copying data between CPU and GPU. Although this memory is typically faster to use, it is a limited resources. ([10] - 3.2.4) Therefore, it is advised to use for small amounts of data that need to be frequently transferred between CPU and GPU.

3. IMPLEMENTATIONS

This paper explores the different methods of utilizing the GPU within application software. Performance is heavily dependant on the software API used. OpenCL and CUDA are two of the most widely used methods for GPGPU. All performance tests in this paper will be implemented using both methods for comparison.

3.1 OpenCL

The Open Computing Language (OpenCL) [4] is an open-standard, cross-platform, heterogeneous programming framework. It provides a single abstract programming model that developers adhere to. Hardware vendors that support OpenCL translate the abstract model to match the particular architecture of their devices. OpenCL currently has support ranging from FPGA, DSP and, of interest to this study, GPUs. OpenCL version 1.2 was used across the benchmarking results collected here, as the lowest denominator supported by NVIDIA, AMD and Intel.

3.2 CUDA

CUDA [14] is NVIDIA's propriety, parallel computing platform for supporting their own GPUs for general compute. CUDA is widely used, including in research and academic studies. In research applications, it can be acceptable to constrain to specific hardware, as is the case with CUDA. However, in industry and commercial environments, this becomes more problematic, as applications implemented using CUDA would not be compatible with machines that have AMD GPUs.

4. BENCHMARK METHODOLOGY

In this section, the benchmarking methodology and tests are described. The system specifications are outlined briefly. The general performance benchmarks are listed, which expose results for GPGPU tests in general. After these, the real-time digital audio tests are defined. These tests aim to identify the limits on the buffer size for example cases, that are representative of typical real-time audio processing scenarios. Harris in [5] explains how to accurately benchmark CUDA applications, OpenCL specific profiling is covered in [15]. The methodology followed here uses CPU timers for measuring overall times and vendor specific profiling tools for measuring isolated parts of the process. The benchmarking suite is open source and available at `https://github.com/Harri-Renney/ThereAndBackAgain-NIME`

4.1 System Specifications

The specifications for the system on which the benchmarking has been performed for this study is shown in Table 2. Various systems, with hardware from different vendors have been chosen in order to observe the performance in general. The systems include discrete GPUs from AMD and NVIDIA, along with an integrated Intel GPU. Integrated GPUs are closely coupled to the CPU and usually have very fast data transfers between them, using faster memory buses and unified memory space. Discrete GPUs typically communicate over slower memory buses across the motherboard and therefore have a larger initial overhead. However, the trade off is that discrete GPUs are usually much more powerful than integrated GPUs.

4.2 Test Format

The general format of the benchmark tests follows the pseudo code below.

```
void test() {
    prepareTest(hostVariables, deviceVariables);

    if(isWarmup) {
        runTest();
    }
    for(int i = 0; i != numRepeats; ++i) {
        startTime = timestamp();
        runTest();
        endTime = timestamp();
    }
    checkTestResults(testResults);
    cleanup(hostVariables, deviceVariables);
}
```

To begin each test, all preparations and initializing of host and device variables are made. Further, kernel code is prepared if necessary. Both CUDA and OpenCL take considerably longer running kernels and data transfers for the first time. This is as preparations and optimizations are made to increase performance of subsequent execution. For this reason, a warm-up variable has been added to the benchmarks controlling whether a warm-up run executes before profiling starts. The test runs begin by timestamping either side of the test. When the tests are complete, the results are checked to ensure the processing done by the GPU is correct. Using the GPU requires memory allocations which the programmer must manually manage. So to finish the test, all associated memory is deallocated.

4.3 Microbenchmarks

This section lists the microbenchmarking tests covered in this paper. The suite includes further tests that are not described here; we intend to explore these in future work.

- *null kernel* - A minimal test to measure the threshold overhead to execute an empty program on the GPU.
- *cpu to gpu to cpu* - A bidirectional test measuring the round-trip transfer time between CPU and GPU.
- *complex buffer processing* - Applies a triangular smoothing operation ([11] - P.g 34. Smoothing) to the input signal and returns 'smoothed' buffers to CPU. Involves bidirectional transfers and multiple memory accesses in the kernel.
- *simple buffer synthesis* - Generation of a sinusoidal signal at a given frequency, generating sine values that fill the buffer length in parallel. This operation involves only unidirectional memory transfers from the GPU to the CPU, returning synthesised sample buffers.
- *complex buffer synthesis* - The complex buffer synthesis is an application that has a challenging amount of computation, bidirectional CPU-GPU transfers and involves memory management. An application which meets these requirements is a finite-difference time-domain physical model synthesizer [18]. For the full details of the design and implementation, see [13].

Each of the tests are implemented in OpenCL and CUDA. Furthermore, each test is implemented using standard memory buffers and pinned memory. All tests have their total times and specific details measured over 10,000 repetitions. From these repeated results, the average, minimum and maximum buffer times are calculated, along with the maximum and average jitter. The results of the warm-up runs have been recorded and are available in the results database.

4.4 Real-time Digital Audio Tests

The audio buffer size is an extremely important factor for achieving real-time performance with GPU acceleration. This section outlines the real-time tests that aim to identify the limits for the buffer size in the unidirectional and bidirectional cases.

To measure real-time performance, the following limits (Described in Section 1.3) must be satisfied:

1. The maximum acceptable latency for each buffer will be 20ms, though the recommended 10ms is preferred.
2. The target sample rate of 44.1KHz should be satisfied within the other limits, though the recommended 96KHz is preferred.
3. The deviation, or jitter, between each buffer generated should not be greater than $\pm 3ms$, though the recommended range $\pm 1ms$ is preferred.

An enumeration of buffer sizes will be applied in each test to find an approximate range of values the buffer size can comfortably operate in. These tests will be conducted for

Specification	Mid-range Laptop	High-end AMD	High-end NVIDIA GeForce	High-End NVIDIA Titan
CPU	Intel Core i7-8550U	Intel Core it-9800X	Intel Core it-9800X	Intel Core it-9800X
Integrated GPU	Intel UHD Graphics 620	None	None	None
Discrete GPU	AMD Radeon 530	Radeon Pro WX 7100	GeForce RTX 2080 Ti	Titan RTX
CPU RAM	8GB	32GB	32GB	32GB

Table 2: Hardware specification used for benchmarking

unidirectional and bidirectional cases. The tables 3 and 4 in the results Section 5 highlights values in green if in the recommended limit, orange if in the maximum limits and red if outside of the limits.

4.4.1 Total Time

Here is proposed an equation for total execution time in a GPGPU environment to formalise the overhead and limitations:

$$t_{total}(x) = t_{tran}(x) + c(x) + g(x) \tag{1}$$

Where:

- $t\_total(x)$ = The total execution time of x samples for a GPGPU application.
- $t\_tran(x)$ = The transfer time for x samples between CPU & GPU.
- $c(x)$ = The function of processing executed on the CPU.
- $g(x)$ = The function of processing executed on the GPU.
- x = The number of samples in the buffer/vector to be processed.

4.4.2 Baseline Limits

The baseline limits will be the time for the minimum transfer and null kernel execution for different buffer sizes. From here, a limit involving variable computation can be derived. Taking Equation 1 and assuming $c()$ or $g()$ are negligible, the baseline overhead is defined as:

$$t_{total}(x) = t_{trans}(x) \tag{2}$$

4.4.3 Kernel Computation Limits

Once the baseline limit has been measured, the kernel computation time can be calculated. By subtracting the baseline $t_{trans}(x)$ from the full Equation 1, the total computation time $t_{comp}(x)$ remains, see Equation 3. If the CPU compute time $c(x)$ is not considered, just the GPU compute time $g(x)$ remains. By considering this Equation 3 and the baseline overhead Equation 2, an understanding of GPGPU becomes more clear. The baseline overhead serves as an initial cost to be considered. From here, the computation cost involved can be increased within the limits of the application.

$$t_{comp}(x) = c(x) + g(x) = t_{total}(x) - t_{trans}(x) \tag{3}$$

5. RESULTS

As is to be expected, due to all the permutations of configurations, a lot of results have been collected. In this section, we analyse particular highlights of the results, in areas considered most relevant within the audio domain. A collection of all results for those interested can be found at: `https://muses-dmi.github.io/benchmarking/benchmarking_database_there_and_back_again`.

The results considered here have been collected following a conservative approach. This means after every API call to the GPU, an explicit synchronization is made between the CPU and GPU. Further, pinned memory in OpenCL is mapped and unmapped to ensure it is defined even though on many systems this is not necessary. This is important to keep in mind and trivial modifications to the tests would improve performance further. By taking this approach, more confidence can be given to the results knowing that they can be improved.

	GeForce2080_cl			GeForce2080_cuda		
Buffer Length	Total Time	Average Latency	Max Jitter	Total Time	Average Latency	Max Jitter
1	6133.319	0.139	0.741	5676.300	0.128	1.032
2	3053.499	0.138	0.796	2838.170	0.128	0.725
4	1518.143	0.137	0.336	1412.804	0.128	0.669
8	751.078	0.136	0.166	708.566	0.128	0.632
16	378.847	0.137	0.156	378.930	0.137	0.743
32	190.267	0.1375	0.199	183.560	0.133	0.622
64	96.077	0.139	0.170	95.619	0.138	0.646
128	48.746	0.141	0.183	51.526	0.149	0.612
256	24.731	0.142	0.217	27.137	0.156	0.337
512	12.663	0.145	0.192	29.270	0.336	0.474
1024	6.348	0.144	0.164	16.507	0.375	0.490
2048	3.110	0.141	0.152	7.866	0.357	0.340
4096	1.606	0.146	0.166	3.816	0.346	0.496
8192	0.907	0.151	0.158	1.375	0.229	0.251
16384	0.498	0.166	0.160	0.706	0.235	0.264
32768	0.368	0.184	0.186	0.508	0.254	0.252

Table 3: Baseline bidirectional real-time test.

5.1 Minimum GPU Overhead

The minimum GPU overhead involved for the different buffer sizes is a key factor for many applications. If the overhead alone exceeds the requirements, then the GPU will not be appropriate for the task. This means the minimum overhead is a good foundational position to start. The results for executing the null kernel test were $0.002051ms$ on the Radeon 530, $0.000455ms$ on the UHD, $0.009392ms$ on Geforce 2080, $0.011468ms$ on Titan. These are impressively small times, but only execute empty kernels, avoiding critical stages transferring data and processing or synthesis. With the bare minimum results established, the bidirectional baseline test which involves round-trip memory transfers and execution of the null kernel is shown in Table 3. The tests operates at a sample rate of 44.1KHz [1] with measurements taken to calculate the total time to process 44100 samples and the latency and jitter per buffer. These results show that even for a round-trip data transfer with no processing, certain smaller buffer ranges are not practical. Buffer sizes 1, 2 and 4 all have total times above a second for 44100 samples. Therefore, for applications that require single or very smaller buffer sizes, discrete GPUs will not perform sufficiently and the CPU would be the better option. Using Equation 2, the transfer time can be used to demonstrate the minimum GPU overhead at each buffer size. For example, a buffer of 128 is $t_{total}(128) = t_{trans}(128) = 0.141295ms$

5.2 Standard vs Pinned

Figure 4 plots the round-trip data transfer from CPU to GPU and back to the CPU for various buffer lengths in OpenCL, with no kernel executed on the GPU. The solid coloured lines indicate the standard buffer allocation and transfer approach, while the dashed coloured lines represent the pinned buffer memory approach. For all the discrete GPUs, it seems that the pinned memory approach performs better in this test. The smallest difference seen for the GeForce2080 is still ± 0.04 and largest for the Radeon530 is

[1]Results for higher sample rates of 48KHz and 96KHz can be found in the results database.

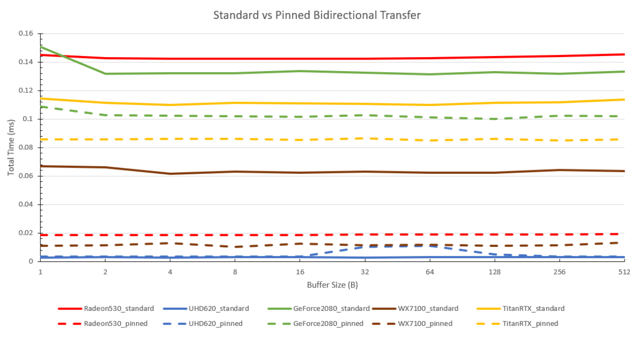

Figure 4: Standard vs pinned bidirectional memory transfers.

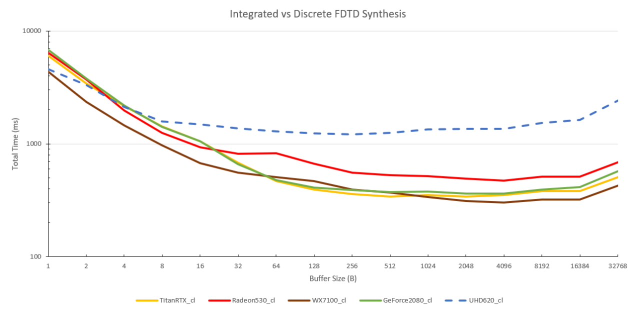

Figure 5: Integrated vs Discrete in bidirectional synthesis.

±0.12. These are significant differences, though it is important to consider no processing is involved in this test and therefore the performance implications during computation as a result of memory choice is avoided. For the integrated Intel GPU, using the standard or pinned approach does not impact performance. This is because the integrated GPU shares its unified memory space with the CPU anyway. OpenCL and CUDA possibly default either approach to the same unified memory approach instead.

	GeForce2080			Radeon7100		
Buffer Length	Total Time	Average Latency	Max Jitter	Total Time	Average Latency	Max Jitter
1	6802.339	0.154	0.958	4372.550	0.099	0.469
2	3790.166	0.171	0.285	2339.507	0.106	0.476
4	2186.325	0.198	0.319	1459.335	0.132	0.387
8	1416.375	0.256	0.321	966.981	0.175	0.546
16	1049.813	0.380	0.395	674.703	0.244	0.617
32	659.516	0.478	0.757	553.152	0.401	0.858
64	478.657	0.693	0.899	508.541	0.737	1.114
128	410.979	1.191	1.178	465.922	1.350	1.544
256	389.878	2.253	2.410	392.696	2.269	2.572
512	373.748	4.295	6.064	370.717	4.261	4.526
1024	377.182	8.572	12.736	338.996	7.704	7.995
2048	363.365	16.516	16.854	309.835	14.083	14.946
4096	361.906	32.900	33.704	302.421	27.492	28.727
8192	393.858	65.643	66.226	322.224	53.704	54.844
16384	414.806	138.268	141.349	320.432	106.810	107.738
32768	570.743	285.371	286.467	427.697	213.848	215.347

Table 4: Physical model synthesizer bidirectional real-time test.

5.3 Integrated vs Discrete

One of the biggest influences on the GPU overhead involved is the type of hardware used. Here, the performance of integrated and discrete GPUs are examined, highlighting where each type of device performs better.

In Figure 5, the bidirectional physical model synthesis is plotted. All the discrete graphics cards have been displayed as solid coloured lines, the integrated graphics card tested is a dashed coloured line. The total time to compute the 44.1KHz of samples has been shown on a logarithmic scale to emphasize the subtle differences. For small buffer lengths, the overhead experienced by most of the discrete GPUs heavily outweighs the benefits and the integrated GPU performs better. However, once larger buffer lengths are used and the transfer overhead reduced considerably, the discrete GPUs take the lead by a large measure. This is expected as physical model synthesis is computationally expensive, and the discrete GPUs have higher computational performance than the integrated GPU. Note that the discrete GPUs settle beneath the 1000ms threshold around buffer length 16. Whilst the integrated GPU is not powerful enough at any of the buffer lengths to successfully compute in real-time.

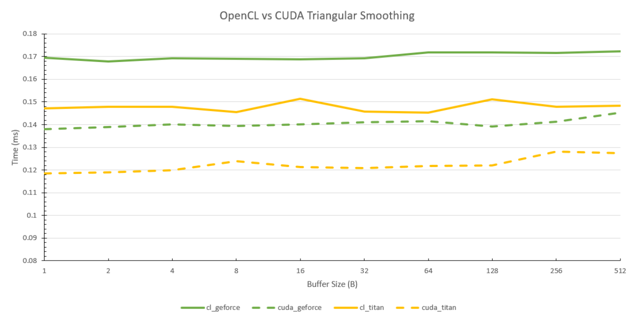

Figure 6: OpenCL vs CUDA for triangular smoothing on various buffer lengths measured in ms.

5.4 OpenCL vs CUDA

The benchmarking paper [3] compares OpenCL and CUDA in general. The paper concludes that for 'trivial' tests they have similar results and in more complicated 'non-trivial' tasks, CUDA appeared to perform better. This section discusses the latest results in 2020 in the audio domain.

Figure 6 plots the results for the complex buffer processing test for different buffer lengths. This test applies a triangular smoothing operation across the signal in the buffers. OpenCL implementations are in solid coloured lines while CUDA is shown in dashed coloured lines. It can be seen that on the NVIDIA GPUs, the CUDA implementations performed better for this kind of task. OpenCL is a defined standard implemented by supporting vendors. Considering NVIDIA develop CUDA themselves, it is possible that they have put more effort into the development of CUDA in comparison to OpenCL. The difference is small, being around $\pm 0.03ms$, though this can make a significant difference under certain circumstances.

5.5 Real-time Performance

The real time performance tests highlight the bidirectional memory transfers for a physical model synthesizer on the GPU. This test involves bidirectional memory transfers, complex and heavy computation and multiple memory accesses from within the kernel. Demonstrating if the GPUs can process a synthesizer like this within the real-time limits will set a high boundary for the GPU environment to prove itself. Table 4 shows the overall time to compute 44.1KHz of samples, the average latency of each buffer and the maximum jitter observed in the test for the NVIDIA GeForce 2080 and the Radeon 7100 in OpenCL. Both implementations seem to have a peak performance around the buffer lengths 2048 & 4096. However, at these sizes, the latency exceeds the recommended $10ms$. The best total performance within the 10ms latency is 512 & 1024. Again however, these buffers have a jitter above $\pm 1ms$. To avoid this, a smaller buffer length of 32 or 64 could be used, which still results in a comfortable half second total time for the whole process to complete.

This test involves the minimal transfer overhead, and the GPU processing. Referring to Equation 1, with buffer size of 128, $t_{total}(128) = t_{tran}(128) + c(128) + g(128)$. By considering the CPU function $c()$ negligible and taking the previously calculated baseline, the equation values consists of $1.191245 = 0.141295 + 0 + g(128)$ and therefore approximately $g(128) = 1.04995ms$. This demonstrates that for a modest buffer size, the minimal overhead is small in comparison to the complex processing that can take place on the GPU. This leaves a lot of room for processing on the GPU to take place, which supports the viability of the GPU in a real-time environment.

6. CONCLUSION

This paper has presented a microbenchmarking suite aimed at profiling GPU performance within digital audio. The benchmarking suite has been used to gather a collection of results across various hardware systems, using both OpenCL and CUDA. From the results gathered, selected sections were highlighted to explore the limitations involved when working within the GPU environment. Buffer sizes dispatched to the GPU for processing is one of the key variables impacting the performance. The general trend observed in the results showed that smaller buffer sizes from 1 to 16 could not meet the sample rate requirement, while larger buffer sizes as high as 32768 down to 512 exceeded limitations for latency and jitter. When comparing different GPU devices, the results showed that integrated GPUs have a significantly smaller transfer overhead between CPU and GPU, this is expected as they share unified memory. The discrete GPUs had a larger initial overhead to transfer data, but performed faster for more complex processing. This reinforces the idea that the integrated GPU is better suited for lighter tasks with less overhead, while the discrete GPUs include a higher initial transfer overhead, but are significantly more powerful. Therefore, assigning them more computationally expensive tasks is recommended. The microbenchmarks test results were also used to compare OpenCL and CUDA on NVIDIA GPUs. On both of the GPUs tested, CUDA appeared to consistently perform better, at least $3ms$ for the triangular smoothing test. It can be speculated that OpenCL support by NIVIDIA is not as well developed as it could be, given that CUDA is their proprietary GPGPU API. The performance benefit from using pinned memory was highlighted, showing a clear advantage for its use when compared with the standard approach. However, performance on pinned memory is dependant on how it is used and has a limited memory size.

7. ACKNOWLEDGMENTS

We would like to acknowledge the consistent support and insight from colleagues in the Computer Science Research Centre and Creative Technology Lab. Special thanks to Sam Hunt, Nathan Renney and Corey Ford. This work was supported by a grant from the HSA Foundation and the UWE 50-50 PhD fund.

8. REFERENCES

[1] J. A. Belloch, B. Bank, L. Savioja, A. Gonzalez, and V. Välimäki. Multi-channel iir filtering of audio signals using a gpu. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6692–6696. IEEE, 2014.

[2] D. Creasey. *Audio Processes: Musical Analysis, Modification, Synthesis, and Control.* Routledge, 2016.

[3] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford, V. Tipparaju, and J. S. Vetter. The scalable heterogeneous computing (shoc) benchmark suite. In *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, pages 63–74. ACM, 2010.

[4] B. Gaster, L. Howes, D. R. Kaeli, P. Mistry, and D. Schaa. *Heterogeneous computing with openCL: revised openCL 1.* Newnes, 2012.

[5] M. Harris. How to implement performance metrics in cuda c/c++. `https://devblogs.nvidia.com/how-implement-performance-metrics-cuda-cc/1`, 2019. Accessed: 2012-10-07.

[6] R. H. Jack, A. Mehrabi, T. Stockman, and A. McPherson. Action-sound latency and the perceived quality of digital musical instruments: Comparing professional percussionists and amateur musicians. *Music Perception: An Interdisciplinary Journal*, 36(1):109–128, 2018.

[7] D. Lavry. Sampling theory for digital audio. *Lavry Engineering, Inc. Available online: http://www. lavryengineering. com/documents/Sampling_Theory. pdf (checked 24.5. 2010)*, 2004.

[8] D. Luebke, M. Harris, N. Govindaraju, A. Lefohn, M. Houston, J. Owens, M. Segal, M. Papakipos, and I. Buck. Gpgpu: general-purpose computation on graphics hardware. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 208. ACM, 2006.

[9] P. Mistry, Y. Ukidave, D. Schaa, and D. Kaeli. Valar: a benchmark suite to study the dynamic behavior of heterogeneous systems. In *Proceedings of the 6th Workshop on General Purpose Processor Using Graphics Processing Units*, pages 54–65. ACM, 2013.

[10] C. Nvidia. Nvidia cuda c programming guide. *Nvidia Corporation*, 120(18):8, 2011.

[11] T. O'Haver. A pragmatic introduction to signal processing. *University of Maryland at College Park*, 1997.

[12] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, and T. J. Purcell. A survey of general-purpose computation on graphics hardware. In *Computer graphics forum*, volume 26, pages 80–113. Wiley Online Library, 2007.

[13] H. Renney, B. R. Gaster, and T. Mitchell. Opencl vs: Accelerated finite-difference digital synthesis. 2019.

[14] J. Sanders and E. Kandrot. *CUDA by example: an introduction to general-purpose GPU programming, portable documents.* Addison-Wesley Professional, 2010.

[15] M. Scarpino. Opencl in action: how to accelerate graphics and computations. 2011.

[16] H. Sutter. The free lunch is over: A fundamental turn toward concurrency in software. *Dr. Dobb's journal*, 30(3):202–210, 2005.

[17] P.-Y. Tsai, T.-M. Wang, and A. Su. Gpu-based spectral model synthesis for real-time sound rendering. In *Proceedings of the 13th International Conference on Digital Audio Effects, Graz*, pages 1–5, 2010.

[18] V. Zappi, A. Allen, and S. Fels. Shader-based physical modelling for the design of massive digital musical instruments. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, page 145, 2017.