GPU audio | ADC24

Accelerated Audio Computing:

Unlocking the Future of Real-Time Sound Processing

November 2024

gpu.audio

Over the past 25 years, our approach to sound has changed little —
from recording and editing music to audio playback across environments like virtual spaces such as games and VR, and real-world setups like venues, concert halls, cinemas, homes, and cars

Let’s take a look into that and find the key reasons why


GPU audio | ADC24
Digital audio computing hardware has made very little progress in computing power over the last 25 years

Pro Audio and Post-Production

AVID HDX

- Up to 256 voiceable tracks per card
- Up to 64 channels
- $4999

Pro Audio and Post-Production

UAD Apollo x16

- Costs $4999
- 450 Mhz
- 2700 Mflops
- 6 chips onboard
- Released in 2010

Media & Entertainment, Professional Audio

Dolby Atmos Processor cp950a

- Costs $3800
- 1 chip onboard
- 64 channels
- Released in 2019

Media & Entertainment, Professional Audio

Audison Virtuoso (ADI Sharc)

- 450 MHz core clock speed
- 5 Mbits of on-chip RAM
- FIR, IIR, and FFT accelerators
- 16-bit wide SDR SDRAM external memory interface
- Year of DSP release: 2012

It is often where professional hardware didn’t receive updates for more than a decade, and what’s more disturbing is that:

- It doesn’t offer much difference for professional user workflows anymore.
  Want to do more? Buy more — ‘simple’
- It doesn’t solve modern professional productivity problems. People spend the same amount of time, and it often doesn’t improve the quality of the results
- Despite being outdated, it remains expensive (sometimes very expensive) and is certainly not affordable for most professionals

The consumer experience hasn’t changed at all. Everyone talks about immersiveness, but do regular people actually feel immersed with standard audio setups? Do they notice any difference, or understand how an "immersive experience" is distinct from a regular one, beyond the marketing language?

In cinema, at concerts, at home, in cars, while playing games?

…And sometimes even new audio DSP hardware has a very little progress in adoption

Media & Entertainment, Consumer Audio

Sony Playstation Tempest DSP

- 1 dedicated CU of AMD RDNA 2 GPU or RDNA 3 GPU, depending on the version
- Price included, $699

…whereas existing PC-based AMD True Audio Next software technology could be easily adopted and used instead of dedicated HW unit

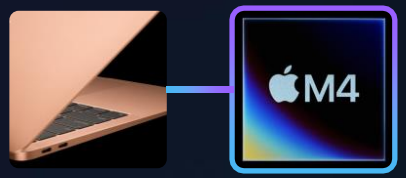
Situations like those mentioned earlier won't change until new methods for utilizing advanced audio computing hardware are introduced

Let’s speak about the new hardware by the way…

GPUs, NPUs, TPUs and DPUs are pretty much everywhere now, from the smallest mobile and embedded to the form-factors we couldn't imagine 5 years ago

Nvidia Jetson Orin

Qualcomm Adreno



M-Silicon/Nvidia/AMD

Nvidia B200

These devices were created to deliver one thing – Accelerated Computing

It began with graphics in the mid-90s. In the 2000s and 2010s, it saw widespread adoption in scientific computing and HPC, and after AlexNet, it sparked extensive use in machine learning.

So how adoption of GPU technology stack is different for Audio?

Can we ‘simply’ use CUDA stack, OpenCL, SYCL or any other way to ‘just program GPUs’ and do things DSPs, FGPAs, CPUs can’t do per same computing cost?

Native implementations can efficiently transfer data and execute a kernel quickly, with numerous studies already conducted on this topic. **Here's one for reference:**

You can also download CUDA (or any other ‘native’ framework SDK) and run some tests quickly and learn the following:

- You can run a dumb kernel within a matter of just **1-3 nanosec**
- Depending on the PCI-e or any other bus generation, you can pack a **decent** amounts data within **10-30 usec data transfer window**
- You can even run **several** graphs of processings within **1-5 msec latency window**

Not bad, isn’t it?

Yet, this won't allow you to create applications because of these fundamental challenges:

Even one kernel execution latency is inconsistent, you could expect **a significant percentile hit with latency drops to 5-20 msec** and there is no way to manage it explicitly using existing frameworks functionality

It is **easy to break overlapping of streams** even if running graphs, which leads to unexpected dramatic bandwidth and latency drops

Add to this:

Over 95% of audio DSP algorithms are inherently sequential by design, **requiring custom parallel implementation.**
For reference, the CuFFT library spans 85 pages and builds on scientific research, including an advanced take on the 'trivial Cooley-Tukey algorithm', which is well-parallelized

Classic GPU frameworks don’t support barrier synchronization for fewer than 32 threads without reducing occupancy. As a result, a **single application often consumes significant GPU resources**, which frequently remain underutilized

Real-world applications consist of hundreds of DSP components running in a mixed parallel/sequential manner. **None of the graph APIs can handle this efficiently without sophisticated device-side optimization and dependency-aware software scheduling**

…and finally add to this tip of the iceberg some standard engineering problems such as:

Supporting your application in cross-platform will basically mean implementing a Cartesian product of CPU_arch x GPU_arch x OS x Programming Model / Software Architecture

The ‘documented behavior / observable behavior / real behavior’ of the GPU code is still normal (yet, undesired) situation, even for best-in-class frameworks and runtime environments

So no wonder that there are almost none of the real commercial applications / product exist that utilize power of GPUs

And there are none of them exist doing it truly real-time, at scale, utilizing a fraction of the GPU resource and without causing any interference with existing GPU workloads, deployed on regular end-user computer systems

We recognized these issues (and many others beneath the surface) and developed the GPU AUDIO Platform. It provides a CUDA-style C++ interface for programming GPUs, along with DSP and Neural library building blocks, enabling the creation of applications, services, and products for nearly any hardware platform and OS

Let’s have a very quick overview on our Platform this year again

GPU AUDIO SDK Workflow Schematics

GPU Audio component (audio processing engine)

- Low-latency scheduler
- Implementation of routines provided by APIs
- Proprietary code, provided as a library

Processor API (interfaces for creating audio processors)

- Open header library
- Provides necessary tools for creating your own audio effects
- Uses GPU Audio engine

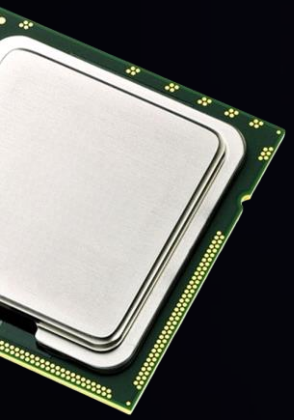
Engine API (interfaces for using audio processors)

- Open header library
- Provides necessary tools for using your own audio effects for processing
- Uses GPU Audio engine

DSP Components Library

- Contains various already implemented filters, partitioned convolution, fft, Neural Network Building Blocks
- Independent of GPU Audio
- Can be used when writing your own audio effects

Cross-platform capabilities



Unified CPU-side interfaces

- Initialization
- Compute Graph Setup
- Port Management
- Memory Management
- Parameter passing

Common device-side C++ style language

- Syncthreads, shared memory, warp communication, etc
- Cache memory operations
- Thread management

Write your code once, and watch as it automatically compiles and deploys seamlessly across multiple platforms

Processor Launcher: Entities

GPU audio | ADC24

GPU Audio

- Scheduler
- Memory Management
- Graph Setup and Validation
- Graph Launcher

Graph Launcher

- Instantiation of Processors
- Creation of Processing Graphs
- Data Transfer Control
- Synchronous and Asynchronous Launch

Processing Graph

- The Processing Graph holds multiple processors and their connections (ports)
- Determines an ideal way of scheduling the graph on the respective hardware, optimizing for number of GPU launches, temporary memory requirements, and latency

Processor

- Core processing functionality of a processor, split into task, blocks, and threads running on the GPU
- Parameter passing control
- Memory management and transfer as needed
- Hints for the gpu audio engine about processing characteristics

*internally it's a special case of DAG

2025 release support will include

Modern desktop-grade chips include:

RTX 4080

9728 cores, 2205 MHz

16GB GDDR6X

$1300

Radeon 7900 XTX

6144 cores, 2500 MHz

24GB GDDR6

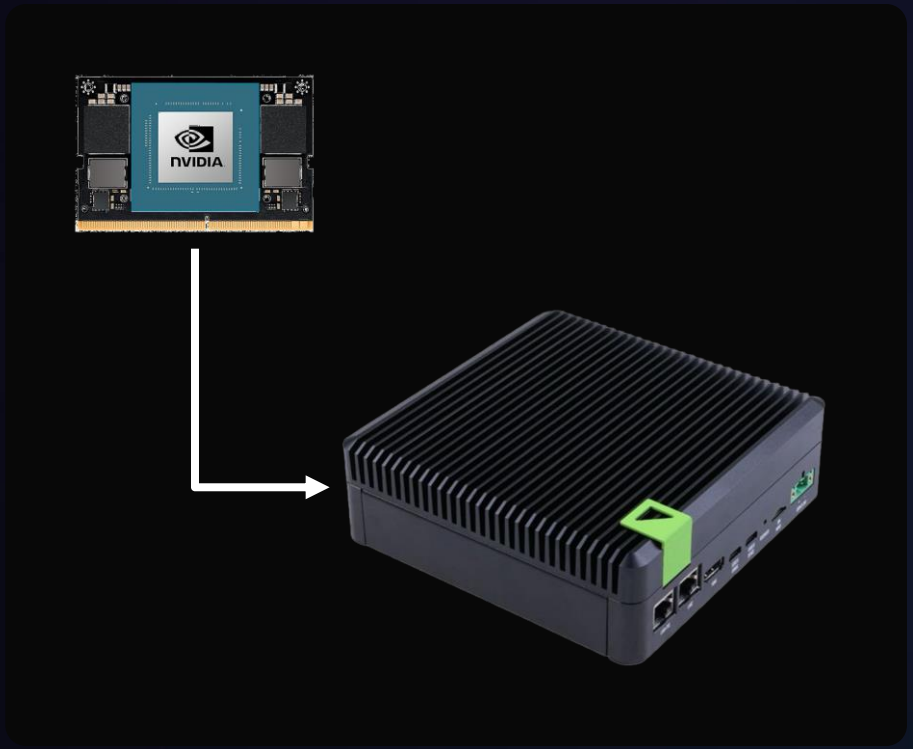
$750

M4 Max

40 cores, 1800 MHz

128GB GDDR6

Included, $3200



Nvidia jetson Orin NX

- 512 Core NVIDIA Ampere, with 16 Tensor Cores
- 1024 Core NVIDIA Ampere, with 32 Tensor Cores
- 70/100 TOPS
- Price starts from $99 for NANO

Nvidia B200

- Built with eight NVIDIA Blackwell GPUs
- GPU Memory 1,440GB total, 64TB/s HBM3e bandwidth
- 72 petaFLOPS FP8 training and 144 petaFLOPS FP4 inference
- Starts from $25 000 per unit

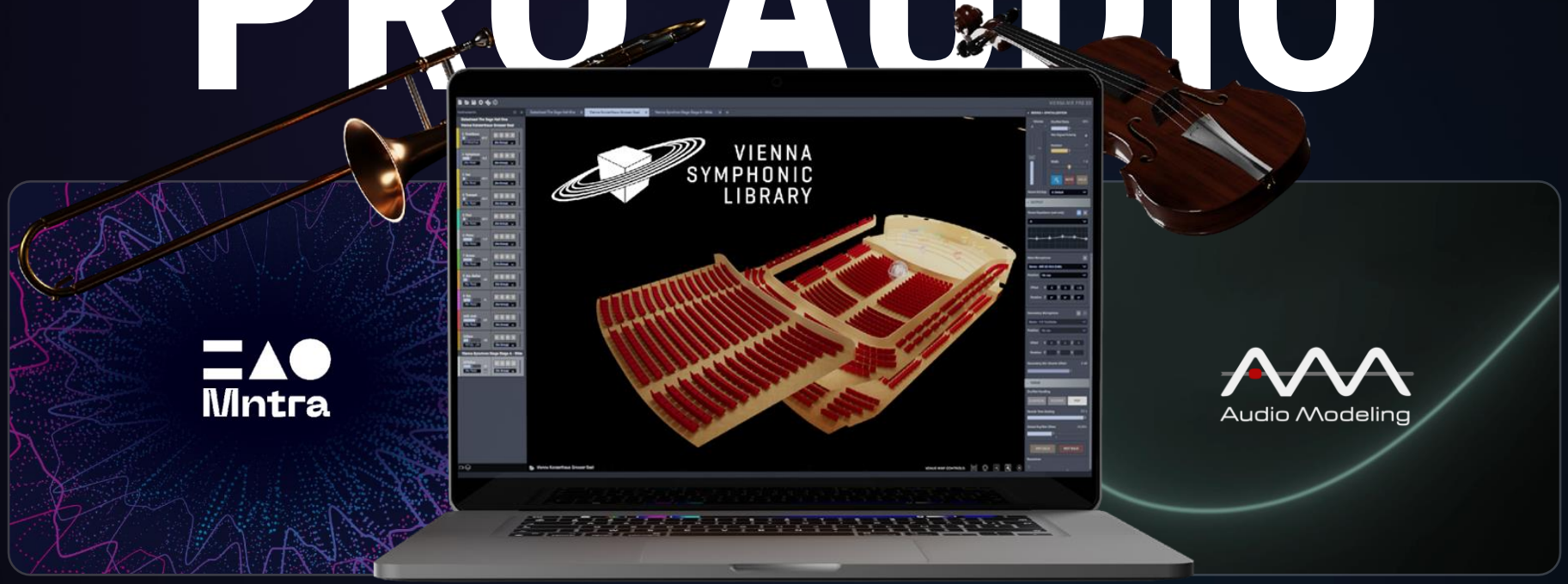
The power of GPU chips can be harnessed effectively from virtually any location where they are embedded

But is there such a demand for the compute?

Novel Use Cases Across Industries

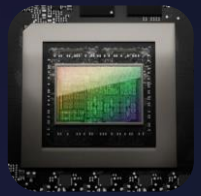
Automotive	Non-immersive sound experience, limited infotainment features	→	Immersive environment and spatial sound experience, rich infotainment scenarios
Pro Audio	Slow rendering, high latency, limited functionality	→	Fast rendering at low latency, next-gen functionality support
Gaming / VR	Pre-recorded non-realistic sound effects and poor ambience capture/playback	→	Real-time ultra-realistic interactive effects and realistic ambience capture/playback
AI / ML	Non-realtime inference prevents streaming AI media generation	→	Real-time inference of small and large models running on any device for streaming gen AI

PRO AUDIO



Gaming

Automotive Audio



5%

of your existing car's GPU device's power to deliver **any software defined functionality for only $25**

Call center solutions

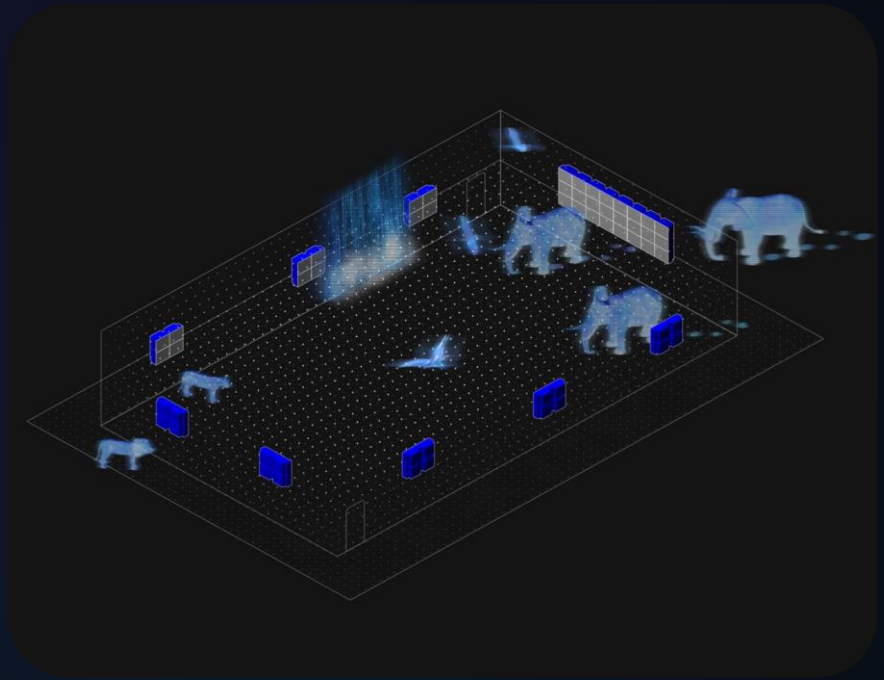
Speech synthesis for voice assistants

Ok, it all looks like ‘we can do more’ or we can just run ‘advanced algorithms’ but how come it can change consumer experiences?

The answer is, it already does!

Holoplot modules use FPGAs to control loudspeaker channels that perform realtime advanced beam steering and wavefront synthesis

… And, venue setup requires scene rendering / mixing solution, which is again, brings additional hardware, server racks, to mix thousands of channels



The total cost of setups based on nextgen audio experience providing solution can easily be millions of dollars

Could that be downscaled? To the concert hall, to the cinema? To the car, home setup? **The answer is yes**

Can it be more affordable? **Totally**

$40K-$60K

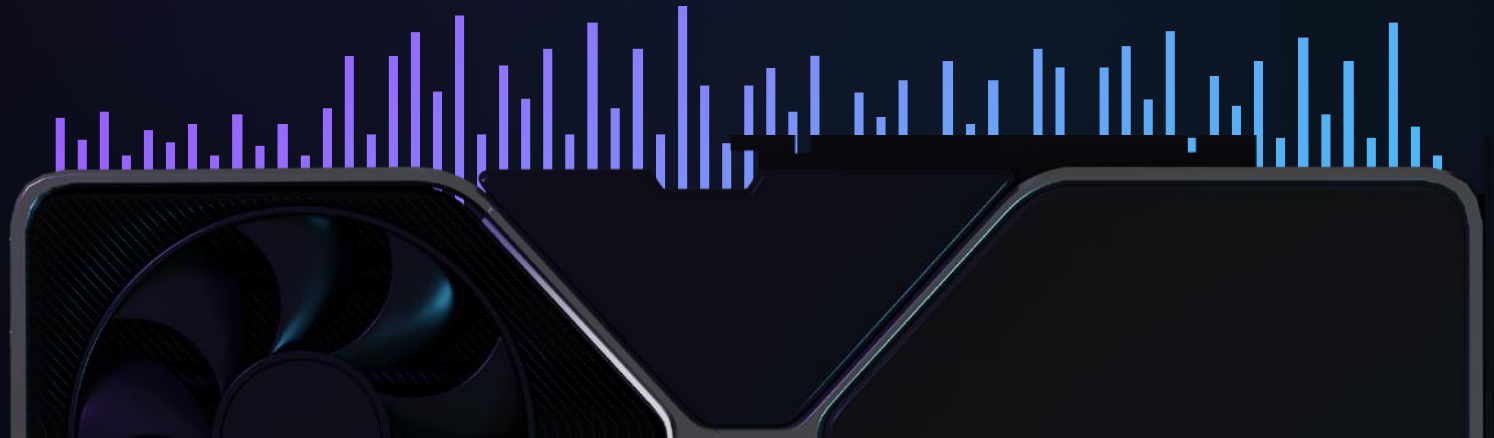
$13K

Accelerated Audio Computing opens the entire plethora of what's possible…

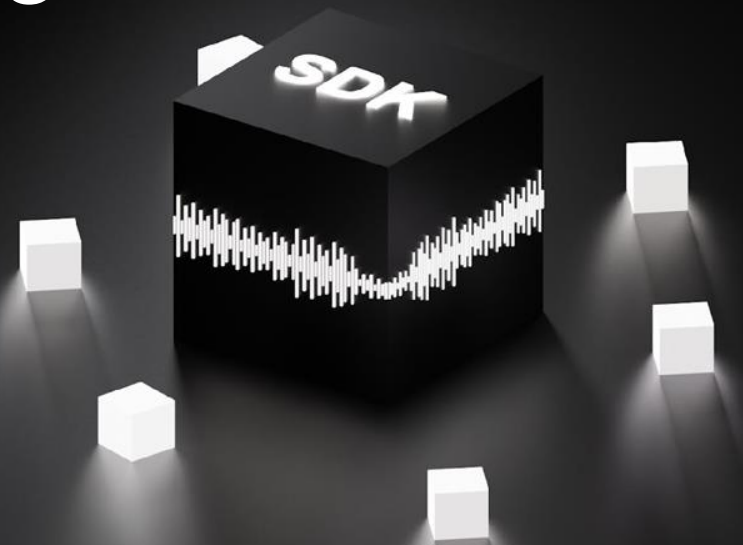
To facilitate this trend, and speed up the development of nextgen professional software and to be able create unprecedented consumer experiences such as Sphere offers, we need to cooperate

Thus, we'd like to establish a format of the open conversations surrounding Accelerated Audio Computing

We Need a Standard



…to solve challenges Such as one we shared during the ADC 2023



Challenges to solve still: Asynchronous process Block call

GPU audio | ADC24

VST

```
ProcessData{...};
Steinberg::tresult PLUGIN_API
process(Steinberg::Vst::Proce
ssData &data);
```

AU

```
OSStatus
ProcessBufferLists(AudioUnitR
enderActionFlags&
ioActionFlags,
                const
AudioBufferList& inBuffer,
AudioBufferList& outBuffer,
                UInt32
inFramesToProcess) override;
```

JUCE

```
void
processBlock(juce::AudioBuffe
r<float>&, juce::MidiBuffer&)
```

All major audio plugin vendors offer blocking process calls in their formats. A major change is required to unlock the full potential of GPU Audio

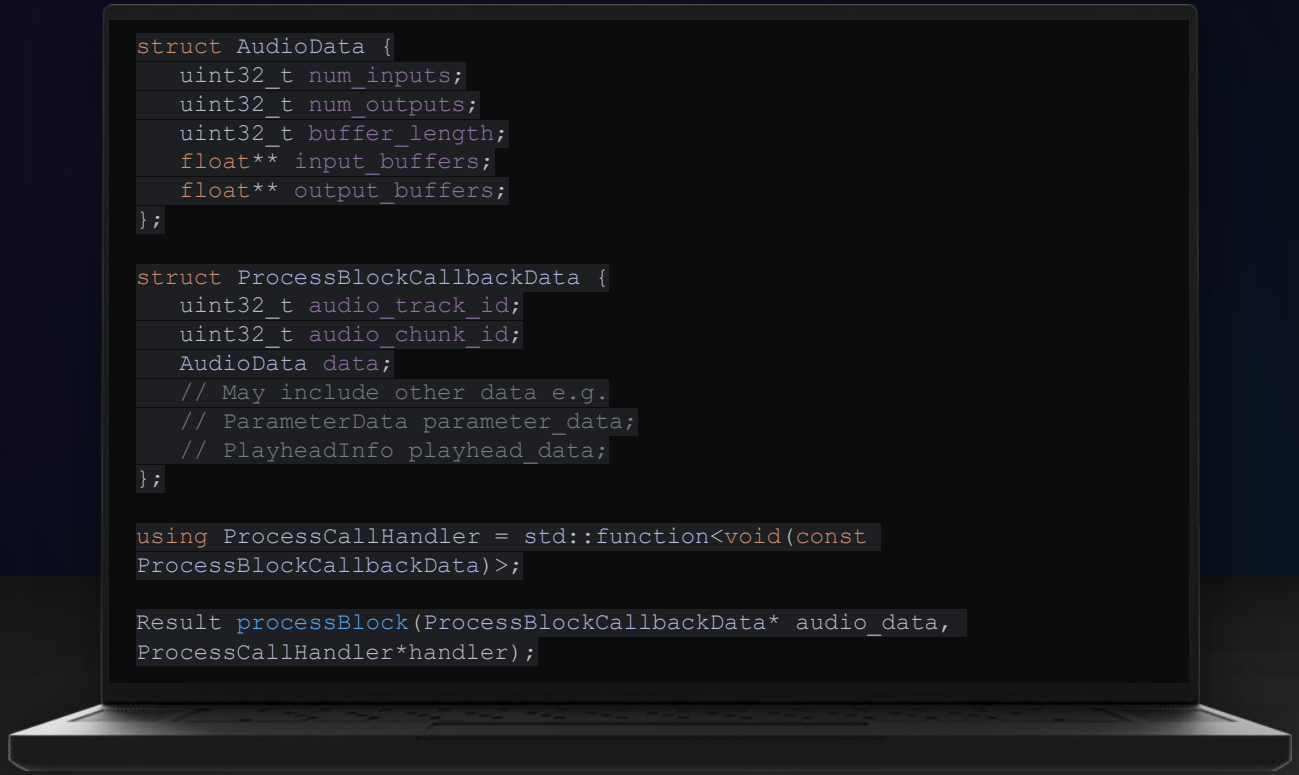
Challenges to solve still: Batch Processing

- The ability to get information regarding where an individual process block originates from would greatly increase utilization of the GPU
- Offline processing, where we could get data faster than real-time and work on a larger set of data
- When all chunks from the tracks have been received, a heuristic on the side of GPU Audio can then determine the best possible launch configuration

Our end-goal

- Being able to tell when we have received all buffers that need to be rendered
- Which tracks and plugin instances they originate from
- The process from which they are coming

Proposed solution: Asynchronous process call and Batch Processing identifiers



```
struct AudioData {
    uint32_t num_inputs;
    uint32_t num_outputs;
    uint32_t buffer_length;
    float** input_buffers;
    float** output_buffers;
};

struct ProcessBlockCallbackData {
    uint32_t audio_track_id;
    uint32_t audio_chunk_id;
    AudioData data;
    // May include other data e.g.
    // ParameterData parameter_data;
    // PlayheadInfo playhead_data;
};

using ProcessCallHandler = std::function<void(const
ProcessBlockCallbackData)>;

Result processBlock(ProcessBlockCallbackData* audio_data,
ProcessCallHandler*handler);
```

Expected result

~0%

CPU usage

Unlocking potential:

- Bringing asynchronous processing will minimize the thread synchronization overhead
- No additional buffering techniques will be needed for plugin developers
- Batch processing would allow better saturation of the GPU's resources, significantly lower CPU usage and faster offline processing

… and to solve many more others:

- Integrations in game engines such as UE or frameworks such as WWise
- Providing a standard build for embedded systems
- Expanding DSP library
- Expanding Neural building blocks

Be the first to join the conversation

And sign-up for future updates at:

accelerated.audio

Next steps are…

First item of the SDK release will be NAM Modeler Workshop

GPU audio | ADC23

Q&A

gpu.audio