AUDIO BENEFITS AND CHALLENGES
OF HETEROGENEOUS COMPUTING

Carl Wakeland
AMD
Principal Member of Technical Staff
OUTLINE

- Overview of current and emergent audio applications
- Characteristics and constraints of PC audio processing
- GPU audio offload research landscape
- Examples of extreme workloads in audio applications
- A brief history of audio on the PC
- Challenges of optimizing audio on APU
- Discussion of an optimized APU real-time audio buffer test program
- Summary and prospects
USE CASES OF AUDIO ON THE PC AND NOTEBOOK

- Gaming Audio
- Home Theater PC
- Virtual Audio Worlds
- Voice Chat
- Transcoding
- Audio Creation and Production
- Music and Video Streaming
- Computer Hearing

PC Audio
CHARACTERISTICS AND CONSTRAINTS OF PC AUDIO PROCESSING

- By nature, audio signal processing is often locally compute-intensive. But, relative to video and graphics, the compute bandwidth of audio at the client level is relatively small.

- Real-time audio often requires short buffers that are particularly sensitive to latency and real-time constraints.
  - Loss of real-time during audio rendering and creation is particularly disturbing to the listener.
  - A brief loss of real-time during video rendering can usually be concealed from the viewer. In audio rendering, this is frequently not the case – loss of real-time causes crackles, dropouts and glitches that are unacceptable to the listener. “The eye integrates, but the ear differentiates…”
  - Thus audio processing must ensure that worst-case computing margins are always met. A fire hose of computing resources that is delivered a microsecond too late is of no use.

- PC audio processing has no dedicated hardware infrastructure that ensures that all latency constraints are absolutely met throughout the signal flow, and relies instead on performance characterization of software applications.
RESEARCH LANDSCAPE OF AUDIO ON GPU

- A survey of the literature finds a number of research papers that have studied the implementation of complex audio and speech algorithms on GPUs.

- A good survey of some recent research and observations:

- Examples of audio algorithms on GPU studied in the literature:
  - 3D audio signal processing
  - Convolution reverbs
  - Complex Speech/Sound recognition
  - Audio source occlusion and ray casting
  - Sound Synthesis (esp. physical modeling)
  - Wave Field Synthesis rendering

- Next, a couple of examples (not exhaustive) of possible growth areas for PC audio.
EXAMPLE OF NEXT GENERATION SOUND PRODUCTION | Wave Field Synthesis

- An array of speakers can provide immersive, “holographic” 3D audio that is independent of the listener’s position, using wavefront reconstruction.

- Based on Huygens's principle. Audio DSP paths for multiple speakers are parallelizable.

- Ambisonics is a complementary approach that provides a similar experience.

- Some references:
  - http://portal.acm.org/citation.cfm?id=1531764&dl=ACM

GNU FDL Source: www.syntheticwave.de/pictures/wave_field_synthesis.swf
GAME AUDIO WANTS MORE SOUND SOURCES AND HIGHER LEVELS OF REALISM

- Quote from FMOD online published gaming audio middleware documentation (for PC):
  
  “On limited sound hardware, which generally only has 32 to 64 voices, it can be challenging to manage your whole game's audio voice allocation when you want to have hundreds or even thousands of sounds playing at once in world (for example in a dungeon there might be 200 torches burning on walls in various places all playing a crackling burn noise).”

- Or imagine a truly immersive reproduction of the sound of a passing train with 100+ point sound sources (wheels, rails, etc), all rendered in a 3D space.

- In the current infrastructure, game audio is required to constrain its processing to tight utilization limits to ensure that real-time is not lost in a shared system. To do this, gaming audio engines must:
  
  - Use clustering and panning of premixed sources, prioritization and dynamic muting when resources are exceeded.
  
  - Scale back effects using reduction techniques such as principal component analysis.

*Used by permission
CHALLENGES OF APPLYING THE RESULTS OF GPU AUDIO RESEARCH TO THE REAL WORLD

- Most published research pertaining to audio-on-GPU examines specific audio algorithms implemented on GPU hardware in isolation, and compares the speedup relative to native processing.

- Many papers report remarkable speedups on specific audio algorithm implementations, from 5x to >10x, for GPU relative to native core implementations.

- Yet in the PC, virtually all audio continues to be implemented using native core processing – Why?

- The reasons are complex and are both technical and historical. It is important to understand the business context as well as the technical aspects. First, the history --
In the early years of PC architecture evolution, general-purpose CPU instruction sets lacked DSP capability. Simple timing generators (e.g., the 8254) were the first “audio accelerators.”

Later, sound cards emerged to enable multimedia and other advanced audio processing in the PC. Sound cards gradually adopted custom DSP architectures that were specialized to the unique demands of audio, and accessed through proprietary drivers. The performance and scaling of these add-in devices advanced independently from general-purpose CPU evolution. This was the early “heterogeneous computing” phase of PC audio.

But sound cards added to the cost of PC systems, and proprietary architectures with minimal programmability limited the ability of ISVs to add value.
NATIVE AUDIO VS. HETERGENEOUS COMPUTE AUDIO | A Brief History

- As the IPC and throughput of CPU architectures grew over time, the major core processor vendors realized that system HW cost could be reduced by absorbing mainstream audio signal processing into the host processor. They added DSP capability (streaming SIMD extensions, aka SSE) to the ISA.

- Microsoft’s OS then adopted an internal audio engine that leveraged these advances, and audio applications began adopting native processing with SSE.

- Intel promulgated High Definition Audio, which was structured for native host-based signal processing and simple, low-cost DMA and Codec hardware.
Finally, starting with Windows Vista®, Microsoft discontinued support of audio offload in the OS. This left the mainstream PC audio industry with the homogeneous “native” computing model as the most commercially viable option.

For most applications of the time, this approach was largely successful, and still dominates PC architecture.
- External audio processors held an edge in quality and realism, but were no longer needed for basic audio.

Cut off from most OS support, and irrelevant to the mainstream, custom sound card audio processing largely receded to niche markets of extreme gaming and pro audio.

But the conversion to homogeneous audio computing in the PC had some unexpected consequences that are now difficult to address, given the current ecosystem.
The homogeneous environment of native computing has resulted in audio being left with an unstable level of computing resources, since it must compete directly with other applications while meeting worst-case constraints. Extreme guardbanding is often needed to ensure reliable operation of real-time audio rendering in mixed-use scenarios.

- While possible for controlled platforms (e.g., embedded DAW hardware), in the end it was not practical for the OS to single out and allocate audio applications a consistent high level of throughput in general, under the ever-changing variety of concurrent, diverse mainstream PC workloads.

Moore’s law ran headlong into thermodynamics, and core processor speeds stopped advancing. Multi-core processing became the new Moore’s law of computing.

But due to a combination of factors, native core processor computing system capacity for audio topped out below a single thread, even as more cores were added.

The general class of real-time audio applications, when competing with other non audio applications and services, cannot reliably expand beyond this single-thread. This is one of the primary motivators of heterogeneous computing for audio.
UNEXPECTED CONSEQUENCES OF HOMOGENEOUS PC AUDIO PROCESSING

- This single thread core mainstream ceiling is expected to remain in place because interconnect, data transport, synchronization and cache architectures that could improve audio on multicore are optimized for a market sweet spot that does not overlap with audio’s needs of streaming workloads having small buffers and needing low latency.

- The competitive market demands of the mainstream cannot accommodate the throughput compromises that would accompany a system-wide tuning of PC architectures toward audio’s specific requirements.

- Thus, audio is a special case of the PC that has a combination of unique design constraints and ROI parameters that do not align well with the large market drivers of multicore PC architecture.
WHY GPU/APU POSSIBILITIES FOR AUDIO HAVE NOT YET BEEN REALIZED

- Since the multicore/interconnect driven PC architecture is topping out for audio, why isn’t the GPU compute shader easily stepping up to fill in the gap?

- First, as the history review showed, the established OS infrastructure on the mainstream PC currently does not support heterogeneous computing for audio.
  - Thus targeted ISV applications or middleware, and dedicated devices beyond the view of the OS are the only possible points of entry for non-native audio computing.
  - For the most part, audio PC ISVs are smaller businesses that have leveraged their limited resources around the mainstream and have limited bandwidth to tackle difficult custom computing paradigms. They need a glide path.
WHY GPU/APU POSSIBILITIES FOR AUDIO HAVE NOT YET BEEN REALIZED

- Secondly, in real systems with a single GPU, audio must compete for resources on the GPU even more so than on processor cores.
  - Multiple GPUs, APU+GPU can help address this.

- Finally, from an architectural viewpoint, audio is not always an ideal fit for the GPU compute shader either…
  - To see this, let’s look at the structure of a typical high-performance PC audio application.
EXAMPLE AUDIO SIGNAL PROCESSING FLOW GRAPH

Source 1 | Decode | Scale | Filter | Reverb | Delay | Mix | Filter | Reverb | Delay | Encode
Source 2 | Decode | Scale | Filter | Reverb | Delay |
Source 3 | Decode | Scale | Filter | Reverb | Delay |
Source 4 | Decode | Scale | Filter | Reverb | Delay |
Source N | Decode | Scale | Filter | Reverb | Delay |

5.33 ms buffer cycle for 256 samples @ 48 kHz
CHALLENGES FOR MAPPING OF AUDIO ALGORITHMS TO GPU COMPUTE SHADER

- As we know, the GPU compute shader is highly efficient at SIMD computing, but inefficient at branching and control flow. It primarily adds speedup to algorithms that are highly data-parallelizable.

- But audio flow graphs are typically tree-structured (previous slide). Audio graphs in games can even have multiple sub-trees that sum to a final mix.

- Only specific DSP functions embedded in an audio flow graph such as FFTs, convolution reverbs, and large FIRs have high parallelization affinity.
  - In isolation, these DSP functions can achieve good speedup in the compute shader, but are frequently embedded within non-parallelizable control flow.

- Frequently, this means an offloaded audio algorithm cannot run exclusively on GPU but must be partitioned between GPU and CPU.
CHALLENGES FOR MAPPING OF AUDIO ALGORITHMS TO GPU COMPUTE SHADER

- As noted previously, the interaction of concurrent, unrelated workloads in a real system can be detrimental to audio performance. For example, in a PC game, the compute and data transport loading of the gaming engine and graphics processing typically limit audio capabilities, even beyond what would be expected from a simple summation of CPU utilization, due to system resource contention – e.g., cache pollution (line-stealing) and latency impacts from resource sharing and buffer copies.

- But the published literature typically does not consider these system concurrency aspects when looking at GPU speedup measures.

- As noted above, moving the audio processing function to multicore is problematic in a mixed workload, as these same resource conflicts affect core utilization, cache utilization, and interconnect bus latency simultaneously and unpredictably, with unacceptable user impacts if game performance or audio processing latency is compromised.

- ISVs and OEMs cannot take the risk, and often must limit real-time audio processing throughput in mixed workloads to less than a single-thread.
CHALLENGES FOR MAPPING OF AUDIO ALGORITHMS TO GPU COMPUTE SHADER

- Partitioning of algorithms between compute shader and CPU incurs latency and synchronization penalties (audio must be buffered and streamed).
- These penalties can offset the speedup of the compute shader, or make the total audio path latency unacceptable.
  - This general assessment is frequently heard from audio ISVs that have tried using GPUs for audio applications in the past.
  - It is often assumed that faster interconnect buses will resolve this issue effortlessly. But newer system bus designs and memory architectures emphasize bandwidth over latency. Historically, faster system buses (such as PCIe® which followed PCI) have sometimes been seen to actually reduce offloaded real-time audio performance rather than improve it, because worst-case latency for small bursts is degraded in favor of higher total bandwidth.
  - However, a buffer passing approach using techniques to be shown next can mitigate some of these issues using OpenCL™ on AMD GPUs and APUs.
OPENCL BUFFER-PASSING AND PROCESSING TEST PROGRAM

- We’ve written a short OpenCL test program (ocLatency) to measure latency of data transfers to and from a GPU shader engine in an audio processing use scenario.

- Test program flow:
  1. Every N milliseconds (processing period):
     a. Write M audio buffers of size P bytes from system memory to shader using DMA
     b. Process L loops of intensive FP math in shader
     c. Read M audio buffers of size P bytes back to system memory
  2. Record the minimum number of complete buffers transferred over DMA to/from shader within the processing period over many thousands of periods (several seconds), and measure max roundtrip delay. Repeat with interleaved (ganged) buffers.
  3. Test with ocLatency running on one GPU, 3D graphics stress program running on alternate GPU.
OPENCL BUFFER-PASSING OPTIMIZATION STRATEGIES (1/2)

Following these optimization strategies when using the OpenCL runtime library provides an important speedup advantage in the audio buffer-passing scenario:

1. Use software pipelining (PIPE_STAGES = 3) to allow read, execution, and write stages to proceed in parallel (writeIdx > execIdx > readIdx). Use non-blocking calls to clEnqueueMapBuffer to hide latency when mapping shader read buffers to system memory. Completion events for read mapping will be saved in event object &ev.

2. Start read buffer mapping before starting the write buffer mapping:

   ```
   pDataOut[readIdx % PIPE_STAGES] = (float *)clEnqueueMapBuffer(cmd_queue, buf[readIdx % PIPE_STAGES], CL_FALSE, CL_MAP_READ, 0, M, 0, NULL, &ev, &err);
   ```

3. Next, map the write buffers (use blocking calls), perform memcpy, and then unmap (for all writeIdx):

   ```
   pDataIn[writeIdx % PIPE_STAGES] = (float *)clEnqueueMapBuffer(cmd_queue, buf[writeIdx % PIPE_STAGES], CL_TRUE, CL_MAP_WRITE, 0, M, 0, NULL, NULL, &err);
   memcpy(&pDataIn[writeIdx % PIPE_STAGES], sysBuffers[writeIdx], M);
   err = clEnqueueUnmapMemObject(cmd_queue, buf[writeIdx % PIPE_STAGES], pDataIn[writeIdx % PIPE_STAGES], 0, NULL, NULL); // This initiates the write DMA to the shader
   ```

4. Next, execute the workload:

   ```
   err = clEnqueueNDRangeKernel(cmd_queue, kworkload[execIdx % PIPE_STAGES], 1, NULL, &global_work_size, NULL, 0, NULL, NULL);
   ```
5. Now, block to ensure that all read buffers are mapped, using the event object from previous slide:

```c
clWaitForEvents(1, &ev);
```

6. Since the read mapping was started at the beginning and ran in parallel, this block should normally be complete by the time it is reached.

7. Finally, copy back the results and free the read buffer mapping:

```c
memcpy(sysBuffers[readIdx], &pDataOut[readIdx % PIPE_STAGES], M);
err = clEnqueueUnmapMemObject(cmd_queue, buf[readIdx % PIPE_STAGES], pDataOut[readIdx % PIPE_STAGES], 0, NULL, NULL);
```

- This approach can be further optimized by interleaving multiple buffers (e.g., from different channels) onto each buffer that is transferred.

- Finally, before starting the streaming process, “prime” the buffers to ensure that they are locally allocated and available. This can be done by issuing the above series of CL runtime calls and memcopies inside a finite loop. This priming ensures that real-time constraints can be met immediately when audio begins streaming.
OCLATENCY TEST SYSTEM CONFIGURATION

- **OS**: Windows® 7 64-bit
- **AMD Phenom™ II X4 955 @ 3.2 GHz**
- **RAM**: 8 GB
- **GPU1** (running Furmark 3D GFx stress)
  - AMD Radeon HD 6800 series
  - 900 MHz GPU clk / 1050 MHz mem clk
  - 1120 unified shaders
  - 1 GB DDR5
- **GPU2** (running oclLatency audio buffer test)
  - AMD Radeon HD 6900 series
  - 800 MHz GPU clk / 1250 MHz mem clk
  - 1408 unified shaders
  - 2 GB DDR5
- **Driver version**: 8.831.2.0 (Catalyst™ 11.3)
- **OpenCL™ version**: 2.4
### SAMPLE OCLATENCY TEST RESULTS

<table>
<thead>
<tr>
<th>Quantum Period (ms)</th>
<th>Buffer Size per channel (bytes)</th>
<th>Processing (loops)</th>
<th>Number of channels per buffer (ganged buffers)</th>
<th>Minimum Valid Buffers</th>
<th>Avg. Valid buffers</th>
<th>Max roundtrip latency (ms)</th>
<th>Avg. Roundtrip Latency (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1024</td>
<td>8</td>
<td>1</td>
<td>19</td>
<td>23.1</td>
<td>0.416</td>
<td>0.206</td>
</tr>
<tr>
<td>5</td>
<td>1024</td>
<td>8</td>
<td>8</td>
<td>104</td>
<td>163.5</td>
<td>0.723</td>
<td>0.232</td>
</tr>
<tr>
<td>10</td>
<td>4096</td>
<td>8</td>
<td>1</td>
<td>32</td>
<td>46.2</td>
<td>0.748</td>
<td>0.211</td>
</tr>
<tr>
<td>10</td>
<td>4096</td>
<td>8</td>
<td>8</td>
<td>208</td>
<td>238.3</td>
<td>0.522</td>
<td>0.325</td>
</tr>
<tr>
<td>24</td>
<td>1024</td>
<td>8</td>
<td>1</td>
<td>104</td>
<td>113.2</td>
<td>0.586</td>
<td>0.210</td>
</tr>
<tr>
<td>24</td>
<td>1024</td>
<td>8</td>
<td>8</td>
<td>632</td>
<td>806.0</td>
<td>0.800</td>
<td>0.235</td>
</tr>
</tbody>
</table>
REMARKS ON TEST RESULTS

- Results are improved over what has been commonly reported in the past. Roundtrip buffer delays are under 1 ms for a non-shared shader core.

- We observed about a 10% penalty on the ocLatency results when running Furmark on the second GPU vs. no stress on the second GPU.

- Running Furmark or any significant workload on the same GPU shader that is running ocLatency results in 0 minimum valid buffers able to complete consistently within the time quanta. For the present time, real-time audio running on a shader core cannot be reliably shared with other processing running on the same shader. Impact of competing memory loads is under evaluation.

- This approach is expected to be extensible to an APU + a discrete GPU, with the shader on the APU running audio, while the discrete GPU processes 3D graphics. In a pro DAW type of scenario, where system and app usage is being managed for intensive audio use, this type of use case is reasonable.

- Many thanks to Cliff Jao, AMD Senior Product Development Engineer, for writing, optimizing and testing ocLatency.
SUMMARY

- There is a great backlog potential of higher-impact audio computing applications waiting for the right computing resources.

- Audio processing has many unique characteristics and constraints that do not match the sweet spot of general-purpose computing architectures. General-purpose PCs do not provide guaranteed real-time processing budgets for audio.

- Heterogeneous computing was the original vehicle for audio processing on the PC. Audio processing was later driven by market and industry forces toward a homogeneous paradigm. In order for audio to break through to new levels of performance, a new heterogeneous computing approach may be needed.

- Many researchers have found promising grounds for tapping GPU compute resources for audio.

- OpenCL on GPU compute can be used to support data-parallelizable audio tasks.

- Improved capabilities for real-time audio buffer passing to/from a shader SIMD engine were tested using carefully optimized calls to OpenCL runtime functions. These tests show promise for exploiting OpenCL for higher levels of audio processing in controlled scenarios.
Disclaimer & Attribution

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. There is no obligation to update or otherwise correct or revise this information. However, we reserve the right to revise this information and to make changes from time to time to the content hereof without obligation to notify any person of such revisions or changes.

NO REPRESENTATIONS OR WARRANTIES ARE MADE WITH RESPECT TO THE CONTENTS HEREOF AND NO RESPONSIBILITY IS ASSUMED FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

ALL IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE ARE EXPRESSLY DISCLAIMED. IN NO EVENT WILL ANY LIABILITY TO ANY PERSON BE INCURRED FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

AMD, the AMD arrow logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc. All other names used in this presentation are for informational purposes only and may be trademarks of their respective owners.

OpenCL is a trademark of Apple, Inc. used with permission by Khronos.

Microsoft, Windows and Windows Vista are trademarks of Microsoft Corp.

PCIe is a registered trademark of PCI-SIG

© 2011 Advanced Micro Devices, Inc.