Realtime Signal Processing on Embedded GPUs

Dr. Matthias Rosenthal
Armin Weiss
Dr. Amin Mazloumian

Institute of Embedded Systems
Realtime Platforms Research Group
Zurich University of Applied Sciences
Motivation

- In Comparison to FPGA / DSP Solutions:
  - Performance Gain: 100x (e.g. Analog Devices SHARC)
  - Fast Development Time
  - System on Single Chip
  - Cost Effective

Nvidia TX Series

CPU
GPU
Signal Processing Applications

Embedded System mit GPU

- CPU
  - Realtime
  - Linux
- GPU
  - CUDA
  - OpenCL

Shared Physical Memory

Cache

Video

Audio

Sensor Data

Processed Data
Challenges

• Short and Deterministic Latency
  • Video: 16.7 ms / Frame (60 Hz)
  • Audio: 0.33 ms / 32 Samples (@ 96 kHz)

• High Input and Output Data Rate
  • Video: 3.0 Gb/s (1080p@60 24-bit RGB):
  • Audio: 5.5 Gb/s (256x7 Channels 32Bit@96 kHz)
Short and Deterministic Latency Variability in GPU Kernel Launch

≈ 99.8% as expected

```
__global__
void identity(float *input, float *output, int numElem) {
    for (int index = 0; index < numElem; index++) {
        output[index] = input[index];
    }
}
```

Outliers

```
numElem = 25
0.1 % - 0.2%
< 0.1%
```
Short and Deterministic Latency Variability in GPU Kernel Launch

Latency $\sim$ Buffer Size

Outliers $\approx 50$ ms
Short and Deterministic Latency Problems

• How to Improve Deterministic Behavior?
  → Solution: Persistent CUDA Kernel
  • Eliminate Launch Time
Short and Deterministic Latency Persistent Kernel

CPU
Host Application

```
... 
while (running) {
    if (new_audio_samples() == true) {
        send_sync_to_GPU();
        wait_for_GPU_sync();
    }
}
...
```

GPU
CUDA Kernel

```
__global__ void audioKernel(…) {
    ...
    // Infinite loop
    while (true) {
        wait_for_CPU_sync();
        // Audio channel processing
        ...
        wait_for_all_threads_to_finish();
        send_sync_to_CPU();
    }
}
```

Sync with Memory Accessible by CPU and GPU
Short and Deterministic Latency
CUDA Memory Comparison Zero Copy $\leftrightarrow$ Managed

Zero Copy

Managed Memory

Not usable for Persistent Kernels
Short and Deterministic Latency Problems

- Memory Accessible by CPU and GPU
  - Use Zero Copy Memory for Synchronization

- What about Parameters?
Short and Deterministic Latency
Infinite Loop: Parameter Communication

CPU
Application

GPU
Audio Processing Thread
Local Parameter Copy

System Memory
Parameter Memory (Zero Copy)

Periodic Update
Slow Access!

Application Writes at Arbitrary Time
• Use Persistent Kernels
• Use Memory Accessible by CPU and GPU
  • Use Zero Copy Memory for Synchronization
• What about Parameters?
  • Speed-up with Local Copy of Parameters
• How to Make CPUs Deterministic?
  • CPU Core Isolation
  • Custom Linux: Initramfs built with Yocto
    • No Flash Access Anymore During Runtime
    • Minimize Influence from other Applications
Challenges

- **Short and Deterministic Latency**
  
  ✅ Video: **16.7 ms** / Frame (60 Hz)
  
  ✅ Audio: **0.33 ms** / 32 Samples (@ 96 kHz)

- **High Input and Output Data Rate**
  
  - Video: **3.0 Gb/s** (1080p@60 24-bit RGB):
  
  - Audio: **5.5 Gb/s** (256x7 Channels 32Bit@96 kHz)
High Input / Output Data Rate
Separate Buffers
High Input / Output Data Rate
memcpy() Measurements

memcpy() Time
4096 bytes @ 48 kHz for 12h on 3 CPUs (A57)

75 % CPU Usage!

Time (µs)
0.5 5..10 10..15 15..20 20..30 30..40 40..50 50..60 60..70 70..80 80..90 90..100 100..200 200..500 500..1000 1000+

# Occurrences
1.E+00 1.E+03 1.E+06 1.E+09

TX1 (Kernel v3.10.96)  TX2 (Kernel v4.4.15)

TX1 (Kernel v3.10.96)  TX2 (Kernel v4.4.15)
High Input / Output Data Rate
Shared Buffer

Signal I/O Card

TX2
- PCIe
- CPU
  - Cache
- GPU
  - Cache

Memory Controller
  - Incl. SMMU

DRAM 8GB

I/O Buffer

memcpy

GPU Buffer
High Input / Output Data Rate

Shared Buffer

TX2

- PCIe
- CPU
  - Cache
- GPU
  - Cache

Memory Controller
  - Incl. SMMU

DRAM 8GB

Shared Buffer

Signal I/O Card
• Existing Solutions for Buffer Sharing
  • GPUDirect RDMA (Remote Direct Memory Access)
High Input / Output Data Rate
GPUDirect RDMA

Desktop
Discrete GPU

- CPUs
- Bridge
- System Memory
- PCIe
- 3rd Party Device
- GPU
  
  \textbf{GPUDirect RDMA}

TX2
Integrated GPU

- CPUs
- GPU
- Memory Interconnect
- PCIe Controller
- 3rd Party Device
- System Memory
• Existing Solutions for Buffer Sharing
  ❌ GPUDirect RDMA → Not Available
  • CudaHostRegister()
High Input / Output Data Rate

CudaHostRegister()
High Input / Output Data Rate
Shared Buffer

- Existing Solutions for Buffer Sharing
  - ! GPUDirect RDMA → Not Available
  - ! CudaHostRegister() → Not Implemented on TX2
  - Video4Linux2 → Userptr Mode
High Input / Output Data Rate
Video4Linux - Userptr
High Input / Output Data Rate
Video4Linux - Userptr

Signal I/O Card

TX2
- PCIe
- CPU
  - Cache
- GPU
  - Cache

Memory Controller
Incl. SMMU

DRAM 8GB

Userptr Mode
Mapped Access

GPU Buffer
• Existing Solutions for Buffer Sharing
  - GPUDirect RDMA ➔ Not Available
  - CudaHostRegister() ➔ Not Implemented on TX2
  - Video4Linux2 ➔ Userptr Mode
Conclusion

- Short and Deterministic Latency
  - Persistent CUDA Kernel
- High Input / Output Data Rate
  - Shared Buffer I/O <-> GPU (based on Video4Linux)
- Feasibility of Low-Latency Signal Processing on GPU

Audio Signal Processing on Nvidia TX2 with 200 Channels @ 48kHz
Questions

Website:  http://www.zhaw.ch/ines/
Blog:     https://blog.zhaw.ch/high-performance/
Github:   https://github.com/ines-hpmm

E-Mail:   matthias.rosenthal@zhaw.ch
          armin.weiss@zhaw.ch
          amin.mazloumian@zhaw.ch