Realtime Signal Processing on Nvidia TX2 using CUDA

Armin Weiss
Dr. Amin Mazloumian
Dr. Matthias Rosenthal

Institute of Embedded Systems
High Performance Multimedia Research Group
Zurich University of Applied Sciences
System Overview
Digital Audio Mixing Console

- Control Unit
- TX2 Audio Processing
- Audio Source
- Audio I/O Card
- Audio Sink
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
Challenges

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

• High Input and Output Data Rate
  • 256 Channels * 7 Inputs * \(\frac{32 \text{ Bit}}{\text{Input}}\) * 96 kHz = 5.5 Gb/s
  • 1080p@60 (24-bit RGB): 3.0 Gb/s
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];
    }
}
```

(90%) 

Pr{T≥t} 

0.1% - 0.2% 

Outliers

numElem = 25

< 0.1%
Short and Deterministic Latency
Variability in GPU Kernel Launch

Latency $\sim$ Buffer Size

Outliers $\approx 50$ ms

Jetson TK1
RT Kernel
identity $<<1,1>>$
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();
    }
}
```
Short and Deterministic Latency
Persistent Kernel: Synchronization

GPU
CUDA Kernel

```c
__global__ void audioKernel(…, volatile int* gpuToken) {
    …
    // Infinite loop
    while (true) {
        wait_for_CPU_sync();
        // Audio channel processing
        …
        wait_for_all_threads_to_finish();
        send_sync_to_CPU();
    }
}
```

wait_for_CPU_sync() {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i == 0)
        while (*gpuToken == NO_AUDIO);
    synchronize_threads();
}

Memory Accessible by CPU and GPU
Short and Deterministic Latency
CUDA Memory Comparison Managed <--> Zero Copy

**Zero Copy**

**Managed Memory**
Short and Deterministic Latency Problems

- Memory Accessible by CPU and GPU
  - Use Zero Copy Memory
- What about Parameters?
Short and Deterministic Latency
Infinite Loop: Parameter Communication

CPU
Application

Writes at Arbitrary Time

GPU
Audio Processing Thread
Local Parameter Copy

Periodic Update

System Memory
Parameter Memory (Zero Copy)

Slow Access!
Short and Deterministic Latency

Conclusion

• Use Memory Accessible by CPU and GPU
  • Use Zero Copy Memory
• What about Parameters?
  • Speed-up due to Local Copy
• How to Make CPUs Deterministic?
  • CPU Core Isolation
  • Initramfs built with Yocto
    • No Flash Access Anymore During Runtime
    • Minimize Influence from other Applications
Challenges

• Short and Deterministic Latency
• High Input and Output Data Rate
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!

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

Audio I/O Card

TX2

PCle
CPU Cache
GPU Cache

Memory Controller
Incl. SMMU

DRAM 8GB

I/O Buffer

memcpy

GPU Buffer

PCIe

Audio I/O Card

memcpy

Memory Controller
Incl. SMMU

DRAM 8GB

I/O Buffer

memcpy

GPU Buffer
High Input / Output Data Rate

Shared Buffer

Audio I/O Card

TX2

PCIe

CPU
Cache

GPU
Cache

Memory Controller
Incl. SMMU

DRAM 8GB

Shared Buffer
• Existing Solutions for Buffer Sharing
  • GPUDirect RDMA
High Input / Output Data Rate
GPUDirect RDMA

Desktop
Discrete GPU

- CPUs
- Bridge
- System Memory
- PCIe
- 3rd Party Device
- GPUDirect RDMA
- GPU Memory

TX2
Integrated GPU

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

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

- Embedded Camera
- Video Input
- CPU
  - Cache
- GPU
  - Cache
- Memory Controller
  - Incl. SMMU
- DRAM 8GB
- Userptr Mode
  - Mapped Access
- GPU Buffer
High Input / Output Data Rate
Video4Linux - Userptr

Audio I / O Card

TX2
- PCIe
- CPU Cache
- GPU Cache
- Memory Controller Incl. SMMU

DRAM 8GB
- Userptr Mode
- Mapped Access
- GPU Buffer
High Input / Output Data Rate
Shared Buffer

- Existing Solutions for Buffer Sharing
  - X GPUDirect RDMA → Not Available
  - X CudaHostRegister() → Not Implemented on TX2
  - ✓ Video4Linux2 → Userptr Mode
Conclusion

- Feasibility of Low-Latency Signal Processing on GPU
  - Professional Audio Mixer with 200 Channels
- Short and Deterministic Latency
  - Persistent CUDA Kernel
- High Input / Output Data Rate
  - Shared Buffer I/O <-> GPU (Video4Linux)
Get started with signal processing on GPU!

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

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