Distributed Audio Processing

Alexander Gustafson
University of Applied Sciences,
Zurich,
Switzerland,
alex_gustafson@yahoo.de

September 3, 2015

Abstract

In modern profesional music studios, the computer has become responsible for tasks that were previously performed by dedicated equipement. Mixing boards, effect processors, dynamic compressors and equalizers, even the instruments themselves, are all available as software. To elivate the processing load on the CPU there is a growing market for specialized DSP coprocessors which can process mutliple channels of digital audio in realtime. These coprocessors are typically connected via Firewire or PCIe and use multiple DSP chips for the processing. This project will examine an inexpensive alternative based on standard Gigabit Ethernet and higher end Raspberry Pi clones.

Declaration

I declare that..

Contents

Chapter 1

Introduction

1.1 Ausgangslage

20 years ago the CPU was just one component of a typical music studio. It was generally used to control and synchronize other equipement such as mixing boards, multitrack recorders, synthsizers and effects processors. Today all of the other equipement exists as software, running in realtime on a CPU host. A typical music studio today is comprised of a CPU, multiple analog to digital inputs and outputs, and some DSP equiped audio processing cards.

Simliar to GPU Cards which can accelerate graphics and visualization applications, audio DSP cards can process multiple streams of high qualtiy digital audio, eliviating the load on the CPU Host Computer. Audio DSP cards typically connect to the CPU via PCI, Firewire, or Thunderbolt. Most vendors of DSP cards offer the possibility to connect several cards in parallel to increase the processing capacity.

Unlike GPU processors however, no open standard has evolved to distribute the load across multiple co processors in the way it has for openGL or openCL. 3D graphics applications profit enormously from the interoperability that openGL offers. No such benefit is available for digital audio applications. Also, unlike OpenGL applications, audio software that is developed to run on an audio DSP card cannot be run on the CPU host. This results in vendor lock-in. The consumer that invests in an audio DSP card and software, must continue to buy from the same vendor in order to build on the the initial investment. If another vendor of DSP hardware creates a superior product, a consumer is unlikely to switch platforms if a significant investment has already been made.

10 years ago this was an acceptable compromise because DSP processors connected via PCIe could provide a significant performance increase. Today however, arm based inexpensive CPUs connected via standard gigabit ethernet could offer a competative alternatvie.

1.2 Ziel der Arbeit

The purpose of this semester project is to design a software based music synthesiser that will run on a network of low cost banana pi devices. Limitations in polyphony will be alleviated by adding a new device to the network. In order to be compatible with existing music recording and composition applications the software will include a VST Plugin that allows music software to send MIDI commands to, and receive audio data from the software. All data communication between the VST Plugin and the Banana PI audio generation software will be handled via ethernet. The VST Plugin will send control data information such as pitch, volume, length, and other expression data. The Banana PI will stream back the generated audio data, as well as necessary metadata so the VST plugin can properly collect and prepare the audio data for the host software.

1.3 Aufgabenstellung

- Anforderungsanalyse mit Priorittsbewertung
- Vergleich von mehreren CPUs und Embedded Systems (Banana Pi, Adapteva, Odroid) hinsichtlich ihrer Nutzbarkeit als Echtzeit Audioverarbeitungsmodule. Mit dem System, das die Anforderungen am besten erfllt, wird die Implementierung gemacht.
- Entwicklung der Audioverarbeitungssoftware in C ++.
- Entwicklung eines VST-Plugins in C++, das als Schnittstelle zwischen gngigen Audio-Software und den Audioverarbeitungsmodule (pkt 3) dient.
- Analyse der Implementierung, um die Ntzlichkeit und Skalierbarkeit zu bewerten. Es ergeben sich dadurch verschiedene Fragestellungen wie z.B. folgende: Kann die Leistung und Polyphonie durch Hinzufgen weiterer Module erhht werden, oder wird der Kommunikations-Overhead schlielich zu gross?

1.4 Resultate

1.5 Glossary of Terms

MIDI

The Musical Instrument Digital Interface specification, first introduced in 1983 defines an 8-bit standard for encoding and transmitting music notes. It's original purpose was to allow one keyboard based synthesizer to controll other music devices. [1] Allthough the specification also describes the hardware and wiring for daisy chaining instruments in a midi "network" most midi communication today transmitted via usb or virtually between audio software.

Zeroconf

Zeroconf is a network service discovery protocl. It is also known as Bonjour, and occasionally refered to as Rendezvous, from the Apple implementations. Howl and Avahi are alternative open source Zeroconf implementations for Linux. Application can use Zeroconf to register or browse for service on a network without the need for a user to provide a specific IP Adress or port number. [2]

AES67

Latency

Chapter 2

Einfhrung ins Thema

2.1 Section Title

Chapter 3

Anforderungsanalyse

3.1 General Application Requirements

The Application has two components, the audio plugin hosted on the main CPU machine, and the processing node which runs on a networked SoC device. The audio plugin forwards midi control and audio data to the processing nodes. The nodes stream the processed audio data back to the audio plugin, which in turn streams it back to the host audio application. The total round-trip time, including processing, should not exceed 10ms. This is the maxium allowed latency for live sound applications. [3]

The applications must be self contained and work without the user needing to install any system libraries, frameworks or servers.[1]

3.1.1 Audio Plugin

The audio plugin has the following requirements:

- runnable as a VST plugin in a standard audio application
- locate and connect with one or processing nodes on the network
- forward midi and audio data from the host audio application to the networked nodes
- receives audio data from the networked nodes and streams this back to the host application

3.1.2 Processing Node

The processing node has the following requirements:

- broadcasts its availability and location on the network

[1] The only exception might be ZeroConf/Bonjour on Linux or Windows. See Appendix

- accepts session initiated by the audio plugin
- accepts controll data from the audio plugin
- processes incoming audio data and midi data from the audio plugin
- streams audio and midi data back to the audio plugin or to the next node in the processing chain

Chapter 4

Implementation

4.1 Architektur

4.2 Developement Environment

4.3 Networking Components

These are general C++ components that are used in both the audio plugin and the processing nodes. They encapsulate the network communicatation handling.

4.3.1 Socket Monitor

The socket monitor class uses a posix select() function to montitor the status of a collection of socket file descriptors. When a file descriptor becomes ready to read from or write to the socket monitor can notify a registered delegate class via a specified callback.

- runs as a thread, blocks on select() until one of the managed filedescriptors becomes ready to read then notifies corresponding listener
- define abstract listener class: FileDescriptorListener
- select can be configured with a timeout, unblocking the call, inorder to check status of application (everything ok? should i shut down? should i update? etc..) but it is more effiecient to use a control socket that can be trigged by the app to unblock the select call when needed.
-

4.3.2 ZeroConf Manager

The ZeroConf Manager class encapsulates calls to the system's bonjour/zeroconf deamon.

4.3.3 AudioStream Manager

This class is responsible for streaming audio data to or from a specified udp port. It implements

Chapter 5

Conclusion

Bibliography

[1] Richard Boulanger and Victor Lazzarini. *The Audio Programming Book*. The MIT Press, 2011.

[2] Stuart Cheshire and Daniel H. Steinberg. *Zero Configuration Networking: The Definitive Guide*. O'Reilly Media, Inc., first edition edition, 2006.

[3] AES Standards Committee Gross, Kevin. *AES standard for audio applications of networks - High-performance streaming audio-over-IP interoperability*. Audio Engineering Society, Inc., 60 East 42nd Street, New York, NY., US., 2013.

Chapter 6

Appendix A

6.1 Compiling the Source Code

Download the Juce C++ Library from GitHub
github.com/julianstorer/JUCE
Additionally download and install the DrowAudio Juce Module Extensions
github.com/drowaudio/drowaudio.git

6.2 Bonjour

Instructions for installing bonjour: