Real Time Systems, FIT, BUT

Real Time Audio Processing

David Mihola
xmihol00

3rd of May, 2024

Abstract

Real time processing of audio data is a long standing field. With the introduction of ever larger deep machine learning (DL) models it is becoming increasingly more difficult to process incoming data in real time with acceptable latency. Therefore, we introduce a C++ framework with interchangeable platform specific optimized DL models for real time local or remote audio processing.

Contents

1 Introduction, Requirements and Specification

The general goal of this project is to develop a real time low latency audio processing framework. The framework shall:

1. be written in C/C++/CUDA,
2. allow to interchange audio processing models via a common base class interface,
3. allow to stream audio from a user node to a compute node, where the compute heavy task will be performed, and then stream the result back,
4. optionally allow the compute heavy processing to be perform locally,
5. have a latency below some threshold (ideally adjustable by the user) which can be occasionally violated, i.e. the system can be soft real time with best effort delivery.

2 Literature Review

Paul Browning discuses how audio from a microphone can be accessed in real time in C++ and implements basic audio processing functions in [1]. Gary P. Scavone implements a C++ class for real time audio input and output in [3]. Nicolas Shu and David V. Anderson implement an audio processing Python package utilizing sockets for distribution of processing load in [5]. Konstantine Sadov, Matthew Hutter and Asara Near discus the trade-off between latency, conversion quality and computational efficiency when processing audio with neural networks (NNs) and they develop an NN of their own with small latency and very good conversion quality in [2]. Sebastian Schaetz and Martin Uecker implement a C++ library for real time multi-GPU data processing with the use of the MPI library in [4].

3 Commonly Used Approaches

There are many public solutions for on device real time audio processing in Python like RealtimeTTS[1] for text to speech synthesis, Real-time-ASR[2] for automatic speech recognition or real-time-voice-conversion[3] for voice to voice conversion. Common approach in these solutions is to chunk the input data stream, perform inference via a DL model on the chunks and then concatenate the results back into an output stream. All of these solutions have to deal with the quality/efficiency trade-off described in [2], i.e. they usually use certain amount of passed chunks when performing inference of the current chunk.

[1] https://github.com/KoljaB/RealtimeTTS
[2] https://github.com/Rumeysakeskin/Real-time-ASR
[3] https://github.com/SolomidHero/real-time-voice-conversion

Solutions written in C++ or other compiled languages are also publicly available, such as DeepSpeech[4] for automatic speech recognition or reim for text to speech analysis and synthesis. Both of these systems are again utilizing DL models that are run locally.
As far as cloud or client/server audio processing goes, the available tools are mostly commercial, e.g. the newly released OpenAI[5] text to speech synthesis or automtic speech recognition by Tencent[6].

4 Methods and Tools

The following listing describes possibly useful tools for accomplishing the above specified task:

- **PortAudio**[7]: a cross-platform, open-source, audio I/O library that enables to write simple audio applications in C/C++.
- **libsndfile**[8]: a C library for reading and writing files containing sampled sound. The library supports a large number of file formats.
- **OpenMP**[9]: a multi-platform API that simplifies multi-threaded programming in C, C++ and Fortran on many shared-memory platforms. It consists of compiler directives, library routines and environment variables that influence run-time behavior.
- **Open MPI**[10]: a library enabling to develop portable, scalable, high-performance, parallel software in C, C++ and Fortran on platforms with distributed memory. The library offers easy to use API for communication between processes via messages.
- **NVIDIA CUDA Deep Neural Network (cuDNN)**[11]: a GPU-accelerated library of primitives for deep neural networks, which provides highly tuned implementations for standard routines such as matrix multiplication, convolution, attention etc.
- **TensorFlow**[12]: an open source ecosystem of tools and libraries for machine learning, which provides Python and C++ APIs. The library also supports CUDA-enabled GPU cards.
- **CppFlow**[13]: a C++ library, which enables simple execution of machine learning models in C++ that were trained with the TensorFlow library in Python.

[4] https://github.com/mozilla/DeepSpeech
[5] https://platform.openai.com/docs/guides/text-to-speech
[6] https://www.tencentcloud.com/products/asr
[7] https://www.portaudio.com
[8] https://libsndfile.github.io/libsndfile/
[9] https://www.openmp.org
[10] https://www.open-mpi.org
[11] https://docs.nvidia.com/cudnn/index.html
[12] https://github.com/tensorflow/tensorflow
[13] https://github.com/serizba/cppflow

5 Proposed Approach

The proposed solution, as visualized in Figure 1, will utilize the MPI library for inter process communication, which can both be run on a single machine or in a client/server like setting. Furthermore, each process can utilize OpenMP for multi-threading. In the case of the client, one thread can be responsible for capturing the user input and other one for presenting the resulting output. The PortAudio and libsndfile libraries will be used for I/O. The number of threads run on the server will vary depending on the used ML/DL model or processing algorithm. This algorithm can utilize the TensorFlow and CppFlow libraries or any other implementation. We will use a simple low pass filter with adjustable delay and jitter to simulate and test possible scenarios of such real time audio processing.

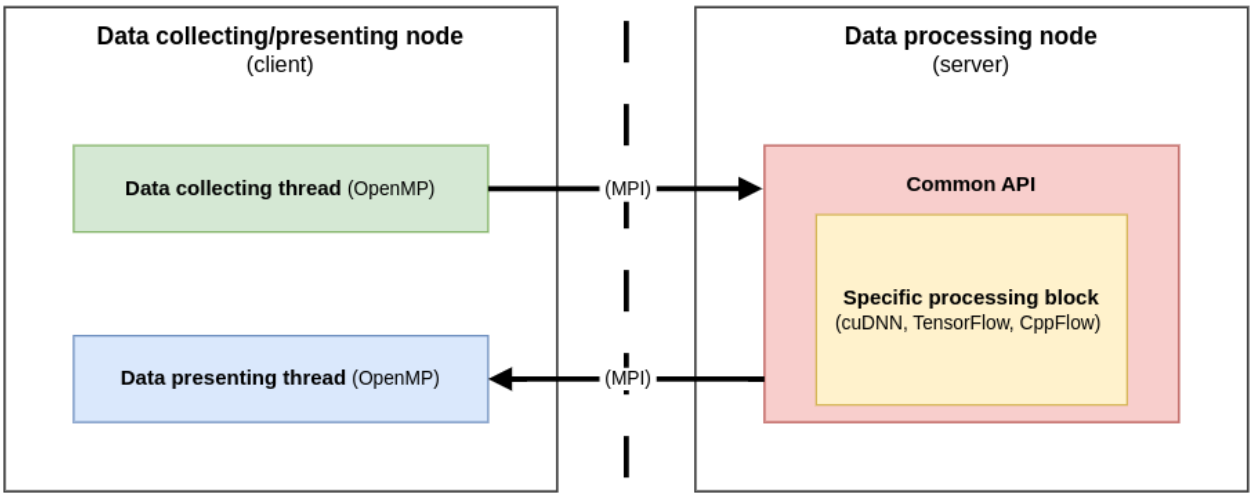

Figure 1: Diagram of the proposed architecture

6 Implementation Details

The architecture of the framework consists of a base `ClientServer` class defining shared variables among the client and server such as sampling rate and window size. This base class is derived by the `Client` and `Server` classes, which implement the client and server sides respectively. Moreover, the `Server` class is an abstract class implementing only the communication with the client. It has a pure virtual method `processChunk()`, which must be implemented by derived classes realizing a specific processing algorithm. Such deriving class is the `LowPassServer` class, which implements the aforementioned low pass filtering.
The framework can operate in four modes selected by command line arguments[14]:

1. direct from microphone to speaker processing,

[14] See `README.md` for the usage details.

2. from microphone to file processing,
3. from file to speaker processing,
4. from file to file simulation of real time processing.

Since there are not any ML/DL models available yet to be plugged in on the server size, the most important mode is the fourth one. Different scenarios, e.g. window size, processing time (delay) or effects of unstable internet connection (jitter), can be simulated and tested prior the development of such models.
The current implementation is available on GitHub[15].

7 Experiments

The `processed_files` directory contains results of simulations with different settings. The files are named with the following pattern `b<number of buffers>_d<processing delay in microseconds>_j<jitter in microseconds>.wav` capturing the used settings. Window size of 33.333 ms (33 333 us) and `test_recording.wav` source file were used for all simulations. The output thread writes zeros to the output file, when data are not available (missed deadline), causing hearable flickering, e.g. `b4_d33700_j0.wav` or `b2_d33600_j3000.wav` are great examples of the phenomenon. We can hear that the flickering appears in waves, which is caused by that the audio processing catches up with the output, when enough zeros are written to the output file. The distorted audio files can be compared to the `no_delay_no_jitter.wav` cleanly processed reference audio file.

[15] https://github.com/xmihol00/MPI_projects/blob/main/audio_processing

References

[1] Browning, P. Audio digital signal processing in real time. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=068defeda78141718232be06e7e92b292d901164, 1997.

[2] Sadov, K., Hutter, M., and Near, A. Low-latency real-time voice conversion on cpu. https://doi.org/10.48550/arXiv.2311.00873, 2023.

[3] Scavone, G. Rtaudio: A cross-platform c++ class for realtime audio input/output. https://www.researchgate.net/publication/228529801_RtAudio_A_Cross-Platform_C_Class_for_Realtime_Audio_InputOutput, 2002.

[4] Schaetz, S., and Uecker, M. A multi-gpu programming library for real-time applications. https://doi.org/10.48550/arXiv.1301.1215, 2012.

[5] Shu, N., and Anderson, D. V. Audiosockets: A python socket package for real-time audio processing. https://arxiv.org/abs/2403.09789, 2024.