Speed-Up of Machine Learning for Sound Localization via High-Performance Computing

Eric Michael Sumner, Marcel Aach, Andreas Lintermann, Runar Unnthorsson, and Morris Riedel

***Abstract*— Sound localization is the ability of humans to determine the source direction of sounds that they hear. Emulating this capability in virtual environments can have various societally relevant applications enabling more realistic virtual acoustics. We use a variety of artificial intelligence methods, such as machine learning via an Artificial Neural Network (ANN) model, to emulate human sound localization abilities. This paper addresses the particular challenge that the training and optimization of these models is very computationally-intensive when working with audio signal datasets. It describes the successful porting of our novel ANN model code for sound localization from limiting serial CPU-based systems to powerful, cutting-edge High-Performance Computing (HPC) resources to obtain significant speed-ups of the training and optimization process. Selected details of the code refactoring and HPC porting are described, such as adapting hyperparameter optimization algorithms to efficiently use the available HPC resources and replacing third-party libraries responsible for audio signal analysis and linear algebra. This study demonstrates that using innovative HPC systems at the Jülich Supercomputing Centre, equipped with high-tech Graphics Processing Unit (GPU) resources and based on the Modular Supercomputing Architecture, enables significant speed-ups and reduces the time-to-solution for sound localization from three days to three hours per ANN model.**

I. Introduction

The Acoustic and Tactile Engineering Lab (ACUTE) of the Icelandic EuroCC National Competence Center (NCC)[1] for Artificial Intelligence (AI) and High-Performance Computing (HPC) performs research and product development for societally relevant challenges in many applications together with its European partners (e.g., project *Sound of Vision* won the Tech for Society award in 2018[2]). This includes the development of wearable assistive devices for visually impaired persons, cochlear implant recipients, and solutions for delivering accurate virtual acoustics (i.e., sounds generated by computers) as realistically as possible and on multi-channel recording/playback.

The lab thus creates and uses large quantities of data; one particular challenge in virtual acoustics is *'sound localization'*, the ability of humans to determine the source direction of sounds that they hear [1]. Several lab members use various AI and Machine Learning (ML) models, such as an Artificial Neural Network (ANN) [2], with audio signal datasets to develop emulations of human sound localization. In addition to identifying the relevant features of these audio signal datasets, another challenge is to determine the right ANN model architecture and the best set of hyperparameters to obtain good model accuracy.

This paper explicitly describes the ANN models' speed-up obtained by porting ML application code to cutting-edge HPC resources, enabling a faster emulation of human sound localization. The original application code used to train the ANN model is a Python program usually executed on laptops of lab members and based on libraries that are not optimized for HPC environments. Consequently, training these ANN models takes several days while cutting-edge supercomputing resources enable a fast training of ANNs via innovative AI libraries such as PyTorch [3]. Porting the application to an HPC environment also significantly improves the overall time to solution for the computationally-intensive optimization of hyperparameters for these models. This paper describes how the application code is ported to the innovative Modular Supercomputing Architecture (MSA) system Dynamic Exascale Entry Platform Extreme Scale Technologies (DEEP-EST)[3] at Jülich Supercomputing Centre (JSC)[4], speeding up ANN model training and hyperparameter optimization while maintaining the model accuracy.

The remainder of this paper is structured as follows. Section II provides a brief introduction to important concepts, i.e. HPC, ANN models with hyperparameter optimization, and sound localization methods. Section III then describes the studied model and the porting strategies employed, including a detailed description of the HPC systems and libraries used. Section IV presents a discussion of the porting results and speed-up achievements. The paper ends with some concluding remarks in Sec. V.

*This work was performed in the Center of Excellence (CoE) Research on AI- and Simulation-Based Engineering at Exascale (RAISE) receiving funding from EU's Horizon 2020 Research and Innovation Framework Programme H2020-INFRAEDI-2019-1 under grant agreement no. 951733. Icelandic HPC Competence Center is funded by the EuroCC project that has received funding from the European HPC Joint Undertaking (JU) under grant agreement No 951732. The JU receives support from the EU's Horizon 2020 research and innovation programme.

Eric Michael Sumner (ems36@hi.is), Runar Unnthorsson (runson@hi.is), and Morris Riedel (morris@hi.is) are with the department of Industrial Engineering, Mechanical Engineering, and Computer Science, University of Iceland, Reykjavík, Iceland

Marcel Aach (m.aach@fz-juelich.de), Andreas Lintermann (a.lintermann@fz-juelich.de), and Morris Riedel (m.riedel@fz-juelich.de) are with the Jülich Supercomputing Centre, Forschungszentrum Jülich, Germany

[1] http://ihpc.is/community

[2] https://soundofvision.net/sound-of-vision-at-ict-2018/

[3] https://fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/DEEP-EST/_node.html

[4] https://www.fz-juelich.de/ias/jsc/EN/Home/home_node.html

978-1-6654-2127-0/22/$31.00 ©2022 IEEE

II. Background

This section provides a brief introduction to important concepts for the experiment: Section II-A describes the architecture of HPC systems in general, Sec. II-B describes relevant ML concepts, and Sec. II-C describes the physical mechanism of sound localization.

A. High-Performance Computing

HPC refers to running applications on computing systems which feature a large number of interconnected processors, and also to software tools which efficiently make use of these systems. With this computing power, it is possible to run tasks much faster than is possible on a single machine with just a few processors. HPC systems are particularly required when processing large datasets with computationally-intensive models.

The goal of an MSA is to improve the overall performance of programs by providing different hardware components for different types of computations. In this case, the supercomputer consists of several computing modules with distinct hardware configurations, connected by an efficient communication system [4]. Neural networks can benefit from such an MSA, e.g. by scheduling the data loading on the Central Processing Units (CPUs) of a compute node and running the compute-intensive training on the corresponding GPUs.

B. Neural Networks and Hyperparameter Optimization

In general, the performance of an ML model strongly depends on its hyperparameters. In the case of ANNs [2], these hyperparameters are related to the optimizer like the batch size, learning rate, or momentum. Parameters of the overall architecture such as the number of layers, neurons per layer, or specifications of the data pre-processing steps are also hyperparameters. Usually, the user needs to set these parameters manually at the beginning of the training.

Hyperparameter optimization describes the process of searching for the hyperparameters of a model which achieve the best possible performance. This search is compute-intensive as it requires a full model training run for many combinations of different hyperparameter values. Several methods are used to reduce these computational costs, such as terminating unpromising training runs early and predicting good hyperparameter combinations via Bayesian optimization [5].

C. Sound Localization

As incoming sound waves interact with the outer ear, internal reflections mutually interfere, modifying the signal received by the eardrum. The human auditory system uses spectral features introduced by this process to determine the sound source's direction [1].

For point sources in the far field, this filtering effect can be characterized as a direction-dependent, linear, time-invariant audio filter, known as the Head-Related Transfer Function (HRTF). As each person's ears have a unique geometry, they likewise have a unique HRTF; determining the relationship between ear geometry and the corresponding HRTF is an area of active research.

HRTF records are stored in the Spatially-Oriented Format for Acoustics (SOFA) [6] file format. As there is no known closed-form representation of an HRTF, this contains impulse responses sampled along elevation, azimuth, and time dimensions. This data array is stored in Network Common Data Form[5] alongside metadata specifying the sampling rates and other details of the data collection process.

III. Experiment

The experiment is based on an existing computational model from our earlier work [7], which emulates human sound localization: Given an audio signal that arrives at the listener's eardrums, it estimates a vector that points to the sound source. This existing model, which was not developed with HPC in mind, serves as the control and is compared to a new implementation for an HPC environment. Section III-A describes the computational hardware used for this experiment. Section III-B describes the data source and how it is used. Section III-C then provides an overview of the model's operation. Finally, Secs. III-D and III-E describe the measures taken to adapt the original application to the HPC environment.

A. Experimental HPC Setup

This experiment leverages the DEEP-EST HPC system at JSC in Germany, which features an MSA-based design. Two different computing modules are used simultaneously to speed up the ANN training. Multiple Extreme Scale Booster (ESB) module nodes with V100 NVIDIA GPUs are used as *'worker nodes'*, with each node processing a different set of ANN hyperparameters. One Cluster Module (CM) node is used as a *'head node'* to coordinate and select the hyperparameters evaluated by each worker node.

B. Data Source and Usage

The data for this experiment come from the 48 HRTF records in the ITA-HRTF dataset [8]. Each of these records contain $360°$ of azimuth data and $160°$ of elevation data, sampled in $5°$ increments. These 2,304 impulse responses have a resolution of 172.3 Hz.

Each neural network is trained against a single HRTF record, to emulate the behavior of a single individual. Half a second of white noise is convolved by each of the impulse responses and the resulting waveforms are presented to the feature extraction stage of the model. For evaluation, 20% of the waveforms are randomly reserved, and the remaining 80% form the training set. The subtended angle between the true source direction and the network's prediction is calculated for each waveform in the evaluation set, and the 95^{th} percentile of these error angles is reported as the model's accuracy.

A hyperparameter search is employed to find a common set of parameters that is suitable for all of the HRTF records. During this search, the performance of 3 HRTFs is directly

[5] https://www.unidata.ucar.edu/software/netcdf/

optimized, subjects 1, 3, and 5: For each trial in the search, the most pessimistic result of these three is considered the true result of the trial. After the search concludes, models are trained for the remaining 45 subjects in the database and analyzed for potential overfitting.

C. Model Operation and HPC Opportunities

Figure 1 summarizes the core model. White noise is convolved with a single direction's impulse response from the HRTF, producing a waveform for analysis. The feature extraction stage then processes this waveform, producing a vector of differences between the left and right channels in three parts: the broadband amplitude difference, band-limited amplitude differences, and band-limited phase delays. This feature vector is presented to the input layer of a feed-forward neural network, and the network's output is compared to the original source direction. These errors are used to update the internal weights of the network, resulting in an estimator for the location of sounds which have been transformed by the HRTF used to train the model.

There are several hyperparameters, highlighted in orange in Fig. 1, without strong justification in the prior literature: The window size used to calculate audio features, the number of frequency bins to include in the feature vector, and the training schedule for the backpropagation algorithm. An Optuna search [9] explores this hyperparameter space to optimize the model for both accuracy and training cost. This produces results comparable to human performance, as reported by Bronkhorst [10].

On a 6-core, 2.6 GHz Intel i7-10750H processor, this process takes 3 days to complete. There are two major opportunities for increased parallelism: The feature extraction and neural-network training processes are both defined in terms of operations on large matrices, which can be calculated in parallel by a GPU. Also, each Optuna trial is calculated in isolation from the others; they can be distributed onto separate compute nodes with minimal communication overhead.

D. Adapting the Model for GPU Computation

The linear algebra and signal convolution routines in the control implementation are provided by NumPy [11]. The CuPy package [12] is a drop-in replacement for NumPy which uses NVIDIA's CUDA[6] system to store and process the data solely within GPU memory. This replacement went smoothly, but there is a maintenance hazard: Each version of CuPy works with only one version of CUDA; if the system CUDA version is changed, CuPy must be manually uninstalled and a different version needs to be installed.

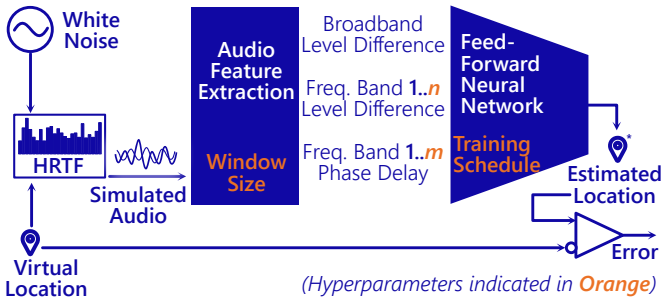

Fig. 1. Overview of the sound localization model.

The ANN feature vector is a set of spectral features calculated over windowed audio. In the control implementation, key parts of this algorithm are provided by the Librosa [13] package. The nnAudio [14] package reimplements these algorithms as convolutional neural networks within PyTorch. nnAudio has a similar, but not identical, API to that of Librosa. The largest change is the use of PyTorch tensors as input and output instead of NumPy arrays. Fortunately, CuPy provides zero-copy conversion routines between its own GPU-based arrays and GPU-resident PyTorch tensors.

There exist two major GPU-based ML libraries to choose from, PyTorch and TensorFlow [15]. As PyTorch is a dependency of this project through nnAudio, it is the natural choice for the main neural network implementation. The ANN in the control implementation is provided by Scikit-learn [16], which is more opinionated than PyTorch. In particular, PyTorch provides a turnkey solution only for the inner details of the training loop– the stopping condition and batching strategy must be hand-written. Additionally, Scikit-learn includes many common training features by default, such as parameter normalization. In order to obtain comparable results, these must be manually enabled in PyTorch.

E. Parallelizing the Hyperparameter Search

Hyperparameter search algorithms have two fundamentally different types of work to do: They must both decide which sets of hyperparameters should be tested and also run the trials to test the selected hyperparameters. These two workloads place different stresses on the underlying hardware. Given the principles of MSAs, they should therefore be assigned to different node types. Optuna, however, is designed for these two workloads to run on the same node.

DEEP-EST provides a Message-Passing Interface (MPI) service [17] to facilitate communication between different nodes. In addition to the standard MPI functions, the MPI for Python implementation [18] also provides facilities to marshal arbitrary Python objects through MPI communication channels. These facilities are used to communicate between two different programs running within the same HPC task.

A cluster node lacking a GPU is responsible for running the Optuna algorithm, selecting appropriate hyperparameter sets to be tested. Each of these hyperparameter sets is then transmitted over MPI to one of a pool of worker processes. The worker process uses the GPU present on an ESB node to train and evaluate the model, and then reports the results back to the Optuna process via MPI. The number of worker processes in the pool is a tunable parameter; 30 were used for this experiment. As each Optuna trial involves the training and evaluation of 3 models, this allowed 10 trials to proceed in parallel.

[6] https://developer.nvidia.com/cuda-zone

IV. HPC Results and Discussion

Both the control and experimental implementations perform an Optuna study consisting of 375 trials, each trial consisting of the training and evaluation of 3 different models. The optimal hyperparameters found by these searches are then used to train models for 45 otherwise unused HRTFs; the accuracy of each model is defined as the 95^{th} percentile subtended error angle. The accuracy distribution of these 45 models show no significant difference between the control and experimental implementations, with a mean of 19° and standard deviation of 4°. This is comparable to human performance on the same task, as reported by Bronkhorst [10].

The time to solution for the experimental implementation is approximately 4% of the control implementation: 3 hours in place of 3 days. In terms of efficiency, the experimental implementation trains and evaluates each model with a single GPU in roughly the same amount of time as the control implementation requires for the same task with a 6-core CPU. This reduced time-to-solution has enabled lab members to expand the hyperparameter search space to, for example, explore other network architectures; this research is ongoing.

The control implementation has two key properties that facilitated the porting process: Most of the computation is delegated to popular third-party modules, which have attracted the development of GPU-based counterparts. Also, the computation is structured as a series of mostly-independent trials which allows a simple but effective multi-node strategy.

The choices of third-party software used for this project are based on a cursory review of the available options, with particular emphasis placed on ease of integration with the preexisting codebase. Other projects may benefit from different choices, such as Ray Tune [19] for hyperparameter searches or TensorFlow for ANN training.

V. Conclusions

This study demonstrates that, with only moderate engineering effort, it is often possible to port CPU-based scientific software to an HPC environment. Algorithms which perform a process of iterative refinement, such as model optimization via a hyperparameter search, are particularly suited to an HPC environment: Multiple speculative trials can be performed in parallel without altering the code responsible for evaluating an individual trial. Performing more trials in parallel will eventually reduce the effectiveness of such algorithms, as there are fewer prior results to base later trials on. In the particular case of a hyperparameter search, however, the 10-way parallelism used in this experiment appears to be significantly below this theoretical limit.

This trial parallelization can be effectively combined with other strategies to speed up individual instances of the computation. This includes not only replacing third-party packages with GPU-based implementations, as demonstrated here, but also the incorporation of packages which can spread individual computations across multiple compute nodes. One such package is Horovod [20], which uses multiple nodes to speed up the training of a single ANN.

By utilizing these two classes of improvement, the time to solution for a particular problem can be greatly reduced. This, in turn, allows scientists to test a larger number of more complex hypotheses in a given amount of time, improving the overall quality of results.

References

[1] J. Hebrank and D. Wright, "Spectral cues used in the localization of sound sources on the median plane," *The Journal of the Acoustical Society of America*, vol. 56, no. 6, pp. 1829–1834, 1974.

[2] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," tech. rep., California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[3] A. Paszke, S. Gross, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, pp. 8024–8035, 2019.

[4] E. Suarez, N. Eicker, and T. Lippert, "Supercomputer Evolution at JSC," vol. 49 of *Publication Series of the John von Neumann Institute for Computing (NIC) NIC Series*, (Jülich, Germany), pp. 1 – 12, 2018.

[5] S. Falkner, A. Klein, and F. Hutter, "BOHB: Robust and efficient hyperparameter optimization at scale," in *Proceedings of the 35th International Conference on Machine Learning*, pp. 1436–1445, 2018.

[6] *AES Standard for file exchange - Spatial acoustic data format*. AES69-2020, Audio Engineering Society, Inc., 2020.

[7] E. M. Sumner, R. Unnthorsson, and M. Riedel, "Replicating human sound localization with a multi-layer perceptron." Submitted to *19th Sound and Music Computing Conference*, Saint-Étienne, France, 2022.

[8] R. Bomhardt, M. de la Fuente, and J. Fels, "A high-resolution head-related transfer function and three-dimensional ear model database," *Proceedings of Meetings on Acoustics*, vol. 29, no. 1, p. 050002, 2016.

[9] T. Akiba, S. Sano, *et al.*, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, (New York, USA), p. 2623–2631, 2019.

[10] A. Bronkhorst, "Localization of real and virtual sound sources," *Journal of the Acoustical Society of America*, vol. 98, 11 1995.

[11] C. R. Harris, K. J. Millman, *et al.*, "Array programming with NumPy," *Nature*, vol. 585, pp. 357–362, Sept. 2020.

[12] R. Okuta, Y. Unno, *et al.*, "Cupy: A numpy-compatible library for nvidia gpu calculations," in *Proceedings of Workshop on Machine Learning Systems (LearningSys) in the 31st Annual Conference on Neural Information Processing Systems (NIPS)*, 2017.

[13] B. McFee, C. Raffel, *et al.*, "librosa: Audio and music signal analysis in python," in *Proceedings of the 14th python in science conference*, vol. 8, pp. 18–25, Citeseer, 2015.

[14] K. W. Cheuk, H. Anderson, *et al.*, "nnaudio: An on-the-fly gpu audio to spectrogram conversion toolbox using 1d convolutional neural networks," *IEEE Access*, vol. 8, pp. 161981–162003, 2020.

[15] M. Abadi, A. Agarwal, *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[16] F. Pedregosa, G. Varoquaux, *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[17] *MPI: A Message-Passing Interface Standard*. Version 4.0, Message Passing Interface Forum, 2021.

[18] L. Dalcín, R. Paz, and M. Storti, "Mpi for python," *J. Parallel Distrib. Comput.*, vol. 65, p. 1108–1115, sep 2005.

[19] R. Liaw, E. Liang, *et al.*, "Tune: A research platform for distributed model selection and training," *arXiv preprint arXiv:1807.05118*, 2018.

[20] A. Sergeev and M. D. Balso, "Horovod: fast and easy distributed deep learning in TensorFlow," *arXiv preprint arXiv:1802.05799*, 2018.