The Sound of Computing

Bachelor Thesis

University of Basel
Department of Mathematics and Computer Science
HPC research group

Examiner: Prof. Dr. Florina M. Ciorba
Supervisor: Thomas Jakobsche

Maarten Schenk
maarten.schenk@unibas.ch

15. February 2021

Abstract

In High Performance Computing (HPC) there is a lot of monitoring data that needs to be analysed and understood. Researchers looking at monitoring data want to understand and compare their applications, as well as identify problems that degrade performance. In order to achieve this, data is usually visualised. An alternative approach is sonification, which refers to turning data into sound. The aim of this work is to develop a custom solution for sonification of HPC data and showcase the possibilities of insights which can be gained. The custom solution uses Python which handles the data processing, sonification and synthesis of the data and uses the Musical Instrument Digital Interface (MIDI) through a Python package, but also provides the possibility to use an open-source synthesiser. It also provides various settings inside a custom user editable configuration file regarding the data selection and sonification parameters. Experiments are conducted to show the functionality of the custom solution. The findings show that sonification can complement traditional visualisation approaches and determine the strengths, weaknesses, and limitations of the custom solution. In conclusion, the custom solution can turn data into sound, but the experiments unveiled that there is room for optimisation. Ideas for these optimisations and future work are discussed.

Contents

Chapter 1

Introduction

1.1 Motivation

In today's world we deal with a lot of data, also in science. High Performance Computing (HPC) is no exception. In HPC there is a lot of performance, diagnostics, and scheduling method data. Traditionally, this data is analysed and visualised. By looking for alternatives to complement the established ways of dealing with data, the idea to turn data into sound came up.

In general, turning data into sound is not a new idea. It is already used in many applications where immediate feedback is necessary or helpful, e.g. Geiger counter, sonar or volcanic activity detection. In HPC, immediate feedback could also be helpful to hear performance drops, load balancing issues, resource bottlenecks, idle times or other metrics. In general, this procedure is called sonification. Sonification is defined as a method to convert information into instrumental sound. The benefits of this procedure are that the auditory perception is able to differentiate between pace, amplitude, frequency and space [1]. This makes sonification a good complementation to visualisation, as it provides an additional perception level.

1.2 Aim

The aim of this work is to gain insight into sonification in HPC, as well as to improve the understanding of HPC operations and find new possibilities of researchable topics. Specifically, to find similarities or dissimilarities between various applications with different executions, number of computing nodes and problem sizes. The focus will be set on the user section from the /proc/stat/ metric, but there are many other possibilities to explore. The /proc/stat/user metric is the CPU's normal processes executing time in user mode, which is typically measured in hundredths of a second.

Additionally, this work aims to be a proof of concept for sonification in the field of HPC and wants to show that insights can be gained if sonification is used on HPC data.

1.3 Problem

Currently there are no widely used solutions that provide all needs which HPC data sonification require. There are many approaches which solve the sonification part, but do not provide out of the box features like multitrack files, intuitive instrument selection, a synthesiser and/or deal with multiple data sources (e.g. how is an average built over performance data? How are different data sources mixed in a meaningful way?). This work will include a custom software solution depending on the used data.

1.4 Solution

To achieve the defined aims, this work will be using the provided data from Ates et al. (2018) [2]. For the sonification of this data, a Python solution with data processing, sonification and a synthesiser will be presented. Moreover, various experiments are conducted to find answers to the above stated questions about similarities and dissimilarities regarding the chosen metrics. In particular, this work wants to explore the following questions based on the software solution:

- Is it possible to hear the differences between different applications?
- Does the problem size have an impact on the sonification?
- Do the nodes sound differently from data of the same run?
- Are runs very similar to each other or must they be handled otherwise?
- Is it possible to use multiple tracks for data (e.g. make all nodes hearable at the same time)?

- Is it possible to feed multiple data sources to a track and still be able to hear similarities or dissimilarities?

A successful outcome could be relevant for any HPC researcher, to gain new, better or different “tune” on their work. Also, if the HPC research field gains more interest in sonification, the sonification community could also benefit from new viewpoints.

Chapter 2

Related Work

2.1 Sonification

To my knowledge there is no prevalent solution for turning performance data into sound. Turning other data into sound is not uncommon. In his master thesis, Andrea Fasolo Rao (2016) [3] designed and implemented a real-time sonification system of solar wind data. His work mainly explored the relationship between scientific visualisation and data art.

Another work by Wanda Diaz Merced (2012) [4], an astrophysicist at the South African Astronomical Observatory who has lost her vision, found in collaboration with programmers a way to continue and further explore her work through sonification. They developed a Java based solution which is designed to analyse two-dimensional data and demonstrated the benefit of their sonification technique with X-ray astronomy and solar data.

NASA uses sonification for measurement data of satellites. Robert Alexander (2014) [5], a graduate student, worked with the sun scientists at NASA for turning data from wind satellite into sound. With their method "bird's ear view" they were able to reduce playback time by a factor of around 4000, thereby turning solar wind data describing a month into a 10-minute-long audio file. These examples show that sonification can bring insights and is successfully used in other scientific fields.

2.2 Sonification solutions

There are many open-source solutions, some of them are more advanced and require no programming knowledge. Others are more a proof of concept and would need further development. A more professional sonification tool, TwoTone created by Datavized Technologies Inc. [6], is a Java-Script based solution, which can solely be run via a web app in a browser. Because it is open-source, it is also available to be built via the Node Package Manager (npm). TwoTone has a lot of features to customise the output and is also flexible on the input data, it is not tailored to do one thing only, like many other approaches. An alternative approach from Erin Braswell is called sonify [7], which uses Python to turn data into sound. It provides many features like playing directly sound from data inside Python, but it requires Python knowledge to utilise.

2.3 Approach

This work will not be using any pervious built solutions, because it wants to use a straight-forward, reproducible, and experimental approach while maintaining a low number of dependencies. Further it aims to be a proof of concept for HPC performance data sonification. Traditionally, performance data is evaluated by visualisation. Contrary to visualisation, sonification might reveal certain information or pattern that cannot be identified through viewing. Sonification puts HPC data into an unusual format and provides a relatively unexplored technique to convey information and perceptualise data. This work combines the existing sonification concept with performance data produced by applications running on HPC systems.

Chapter 3

Methods

The herby proposed solution works with basic methods in order to achieve understandable and reproducible insights. The focus lies more on proof of concept and proof of gain than on providing a full-fledged industry grade solution.

3.1 Mido and Musical Instrument Digital Interface (MIDI)

To achieve the aim, the programming language Python with the Mido package [8] was used, which gives full access to the MIDI communications protocol. MIDI is not a recorded audio signal, it contains only event messages that specify instructions for the device or application playing the MIDI track. These event messages include a note's notation (start or end of a note), pitch (frequency of the sound), velocity (how forcefully a pitch is created) and the clock signal (pace of the track). The range of the playable notes with MIDI is C^{-1} to G^9 which is denoted in numbers from 0 to 127 for the protocol and physically corresponds to the frequency ranged from 8.175799 to 12543.85 Hz (assuming equal temperament, in which notes are separated by logarithmically equal distances, and a 440 Hz A^4) [9].

3.2 The custom solution - sonificator.py

With the MIDI format as a starting point, a custom Python program "sonificator.py" was developed. The software can convert data to sound. The data is handled based on later explained configuration options. If needed, the data is converted to an average (e.g. when four nodes are selected). Depending on the configuration, differences are calculated and calibrated to a scale of pitches. This mapping of data to notes is based on a normalisation over the MIDI or the selected shorter range of pitches.

3.3 Data

The data processing of the Python program is currently tailored to work with all available data from Ates et al. (2018) [2], but could be extended to work with other data sets. The data from Ates et al. (2018) offers an enormous amount of HPC performance data from 11 different applications with many metrics. For each application they used three different problem sizes and ran it for 30 times on a system with four processing nodes and gathered 721 different metrics. They collected this data because they wanted to know which applications are running on modern supercomputers that are shared among thousands of users. They used machine learning methods to classify known applications and detect unknown applications [2].

3.4 Configuration

To be able to work with such a diverse data set, the Python solution is using configuration files which can be easily customised by end users. A configuration file is composed of different sections, each preluded by a [section] header, followed by key/value pairs separated by a specific string (= or : by default) [10]. In this work, the configuration file starts with a [GENERAL] section, which defines the following settings:

- NAME – sets the folder name for the track(s) and possibly data. This folder will be deleted if it exists before the execution.
- DATA_PATH – sets the path to data.
- DATA_TO_DISC – enables/disables data storage (raw data).

- DATA_TAGS – list of header names for the [“data container(s)”] separated by a space.
- MODE – there are four different modes available.
  - single – allows only one data tag and “sonifies” the data based on the selected [SONIFICATION] settings.
  - multi – allows for any amount of data tags and “sonifies” the data based on multiple settings that are defined inside the [SONIFICATION] and [MULTITRACK] header.
  - difference – requires exactly two data tags and “sonifies” the absolute difference between the data based on the settings inside [SONIFICATION] and the SUBMODE.
  - difference2 – allows for any amount of data tags. This mode determines the average over all input data, calculates the absolute difference between every data input and the average. Afterwards it sums up all differences and “sonifies” the resulting data based on the settings inside [SONIFICATION] and the SUBMODE.
- SUBMODE – this mode is only available for difference and difference2. The sonification in this work is made only over pitch. This mode provides an option for velocity as a sonification argument, which then will always play the same tone but with different velocities.

The [SONIFICATION] section consist of:

- INSTRUMENT – allows the selection of 128 different instruments based on the MIDI standard (a list can be found on `https://en.wikipedia.org/wiki/General_MIDI#Program_change_events`).
- PITCHES – number of playable pitches. MIDI offers 128 different pitches; this parameter shortens the range of pitches to the centre (e.g. a standard piano has only 97 pitches).
- VELOCITY – defines the force which is applied to the instrument (mainly notable in loudness).
- VELOCITY_BANDWIDTH – only available in both difference modes, defines the amount of possible different velocities.

- CUT – decides how data is modified to the same length. If cut is enabled all data is reduced to the shortest data set, if cut is disabled short data sets are extended with zeros to the longest data set.
- DELTATIME – times between MIDI messages (notable in the playback speed).
- MIDITRACK_NAME – the output name of the produced MIDI track.
- FLUIDSYNTH – enables/disables FluidSynth as a synthesiser (FluidSynth must be installed separately.) FluidSynth options are explained in the [FLUIDSYNTH] header section [11].

As mentioned before, there is a [MULTITRACK] section, which only affects the "multi"-track mode. It is composed of two options:

- INSTRUMENT – defines where the instrument should be set, either inside the ["data container(s)"], so that every data set has the option for an own instrument or inside [SONIFICATION] all data sets use the same instrument.
- NORMALIZATION – defines if the data is normalised locally (individually by itself) or globally (over all data sets).

The data set(s) are defined inside ["data container(s)"], the names for them are provided by the DATA_TAGS option from the [GENERAL] section. The options are:

- APPLICATION – all 11 applications provided by the data from Ates et al. (2018) [2] are selectable and shown in Table 3.1.
- PROBLEM_SIZE – the three different problem sizes are denoted as X, Y, Z which corresponds to the difficulty from small to large.
- RUN_ID – defines the selected run (from 0 to 29).
- NODE_ID – defines the selected node (from 0 to 3). It is possible to select all nodes to get an average over all nodes, via a modifier.
- METRIC_ID – defines the metric (721 possibilities). As in the introduction stated, this work will only focus on a single metric regarding the CPU user usage from /proc/stat/.

Table 3.1: Selectable Applications

Application	Description
bt	Block tri-diagonal solver
cg	Conjugate gradient
ft	Fourier transform
lu	Gauss-Seidel solver
mg	Multi-grid on meshes
sp	Scalar penta-diagonal solver
miniAMR	Adaptive mesh refinement
miniMD	Molecular dynamics
CoMD	Molecular dynamics
miniGhost	Structured PDE solver
kripke	Sn transport

- SERIES_LEN – cuts data ends by provided amount.

The options for the FluidSynth synthesiser are defined inside the [FLUIDSYNTH] section:

- SOUNDFONT – path to the SoundFont which should be used to synthesise.
- OUTPUT_NAME – the name of the output file.
- OUTPUT_FORMAT – the output file format, different formats are available (e.g. wav, ogg).
- SAMPLING_RATE – defines the sampling rate (the standard CD rate is 44100 Hz).

The program is able to generate audio files based on these explained configurations, which fully describe the data sets, sonification and synthesiser capabilities. Based on these configurations, this work conducts the following experiments:

- Experiment 1: Compare all 11 applications.
  App [all 11 apps], problem size [Z], run id [0], node id [average over all four nodes]

- Experiment 2: Compare problem sizes X, Y and Z.
  App [selected app], problem size [X, Y, Z], run id [0], node id [average over all four nodes]

- Experiment 3: Compare differences between nodes.
  App [selected app], problem size [Z], run id [0], node id [nodes difference]

- Experiment 4: Compare differences between runs.
  App [selected app], problem size [Z], run id [run difference], node id [average over all four nodes]

- Experiment 5: Demonstrates the multitrack functionality over the four nodes.
  App [selected app], problem size [Z], run id [0], node id [multitrack nodes]

- Experiment 6: Demonstrates the multitrack functionality over all 11 applications.
  App [multitrack over all apps], problem size [Z], run id [0], node id [average over all four nodes]

These experiments aim to show that with this software it is possible to "sonify" HPC performance data in a meaningful way.

Chapter 4

Results

To evaluate the solution, I listened to audio tracks produced by the Python program. Depending on the experiment, the different modes were used to compare tracks and gain meaningful insights about the performance data.

4.1 Experiment 1 - Is it possible to hear the differences between different applications?

The aim of this experiment is to investigate if the 11 applications could be distinguished solely based on the user data from /proc/stat/. To achieve this aim, I used the single mode for each of the 11 applications. Every track is based on the problem size Z, the 0^{th} run and the average over all four nodes.

> *Experiment parameter:*
> *App [all 11 apps], problem size [Z], run id [0], node id [average over all four nodes]*

The experiment showed that it is possible to distinguish some applications. For example, the applications bt, sp and ft are very difficult to differentiate. Others like kripke, mg and cg all seem to have a rhythm and with different pitch bandwidths, which makes them distinguishable. A very special application is CoMD, which is very monotonous and has some anomalies where it goes wild.

4.2 Experiment 2 - Does the problem size have an impact on sonification?

The aim of this experiment is to investigate the impact of the problem size on performance data. To achieve this, I used the difference2 mode to generate a track which solely makes the difference hearable between the problem sizes.

Experiment parameter:
App [selected app], problem size [X, Y, Z], run id [0], node id [average over all four nodes]

Repeated for bt, kripke and CoMD

By analysing and visualising the data there is a strong indication that the problem size does not influence sonification. After listening to three samples of the selected applications, the used method fails to confirm the expectation that only pitches on the lower end of the frequency bandwidth are audible when the differences in the data are small. This is because the data is normalised for the mapping to the MIDI pitches, which skews the data over the 128 pitches and fails to represent the scale of the data. Because the mean difference in all iterations was less than 1%, it is safe to assume that the problem size does not matter. For any further experiment, the problem size will be set to Z. This also supports the sole use of the problem size Z in experiment 1.

4.3 Experiment 3 - Do the nodes sound differently from data of the same run?

The aim of this experiment is to investigate the impact of the nodes data. I want to answer if it matters which node is chosen and if the average of the nodes is suitable as a replacement for the individual nodes. To achieve this aim, I used the difference and difference2 mode to generate tracks that make differences between the nodes hearable.

Experiment parameter:
App [selected app], problem size [Z], run id [0], node id [nodes difference]

Repeated for bt, kripke and CoMD

By analysing and visualising the data there is a strong indication that replacing a single node with the average does not influence sonification. After listening to three samples of the selected applications, the used method fails to confirm the expectation that only pitches on the lower end of the frequency bandwidth are audible when the differences in the data are small. This fails due to the same reason as experiment 2. As in experiment 2, the mean difference in all iterations was less than 1%. Therefore, it is safe to assume that the selected node does not have an influence.

For any further experiment, the average over all nodes will be used. This also supports the sole use of the average in the previous experiments.

4.4 Experiment 4 - Are runs very similar to each other or must they be handled otherwise?

The aim of this experiment is to investigate the impact of the runs data. I want to answer if it matters which run is chosen or if any run is suitable. To achieve this aim, I used the difference and difference2 mode to generate tracks that make differences between the runs hearable.

Experiment parameter:
App [selected app], problem size [Z], run id [run difference], node id [average over all four nodes]

Repeated for bt, kripke and CoMD

By analysing and visualising the data there is a strong indication that the selected run does not influence the sonification. After listening to three samples of the selected applications, the used method fails to confirm the expectation that only pitches on the lower end of the frequency bandwidth are audible when the differences in the data are small. This is also due to the same reason as the two previous experiments failed. The mean difference in all iterations was again less than 1%, it is safe to assume that the selected run does not have an influence.

For any further experiment, the 0th run will be used. This also supports the sole use of a single run in the previous experiments.

4.5 Experiment 5 - Is it possible to use multiple tracks for data?

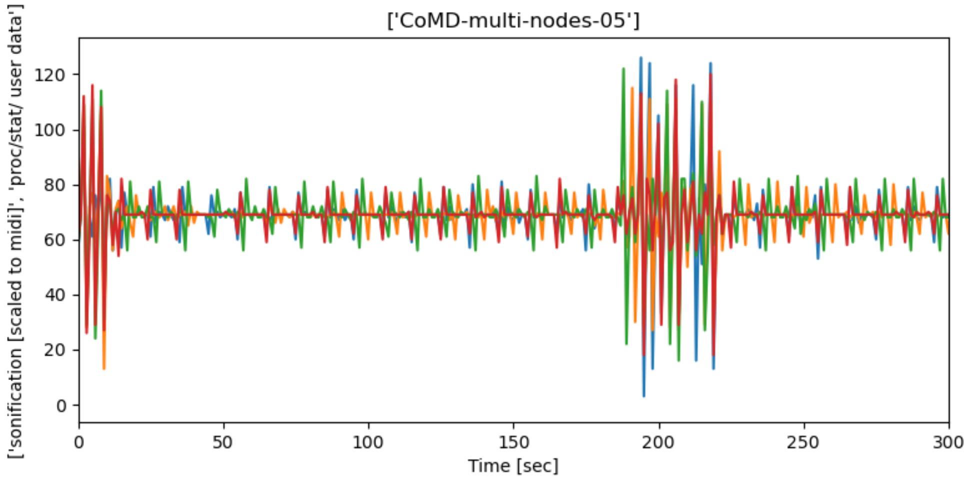
The aim of this experiment is to show the functionality of multiple data sources inside one track. To achieve this aim, I used the multi-mode to generate a track with multiple data sources.

Experiment parameter:
App [selected app], problem size [Z], run id [0], node id [multitrack nodes]

Repeated for bt, kripke and CoMD

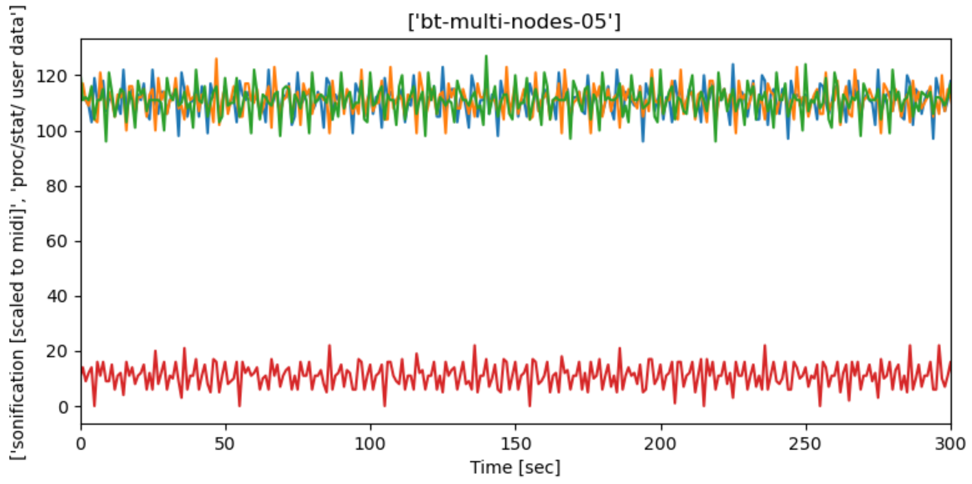
I used the data of the nodes from the same run. Surprisingly, the result was a good method to make differences hearable. Even for an untrained person that lacks a musical ear it is easy to notice when a part plays out of tune. Figure 4.1 shows the data of all four nodes scaled to MIDI notes.

Figure 4.1: CoMD all four node data as multitrack.



In the case of the bt application, there are three nodes playing very high pitches, while the last node is only playing low pitches. Figure 4.2 shows a prime example for a node playing out of tune.

Figure 4.2: bt all four node data as multitrack.



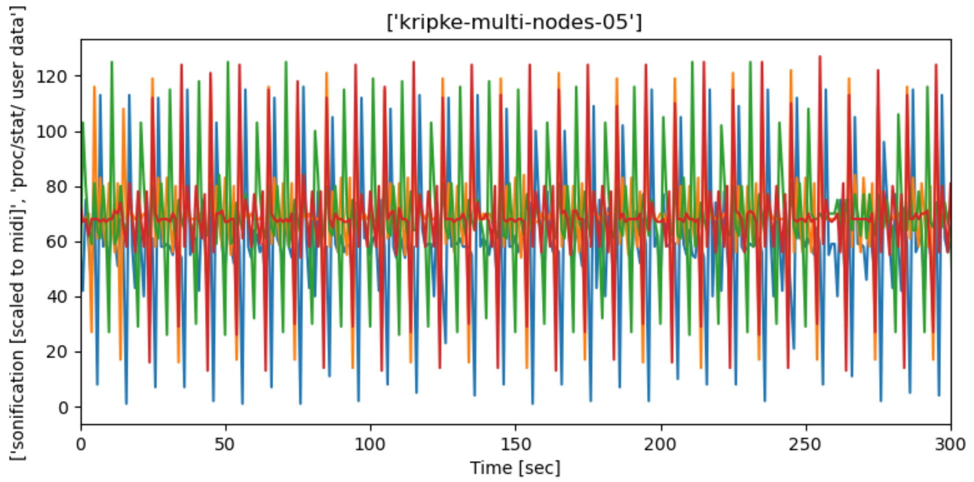
When using the kripke application, there is a broader variation of tunes. This is explained by the fact that the min, max and mean are all in less than 1% of the data scale, which gets distorted by the same reason as in the previous experiments, as shown in Figure 4.3.

4.6 Experiment 6 - Is it possible to feed multiple data sources to a track and still be able to hear similarities or dissimilarities?

The aim of this experiment is to show an overview over all 11 applications and their load in comparison. To achieve this aim, I used the multi-track mode to generate a track from all 11 applications data sources.

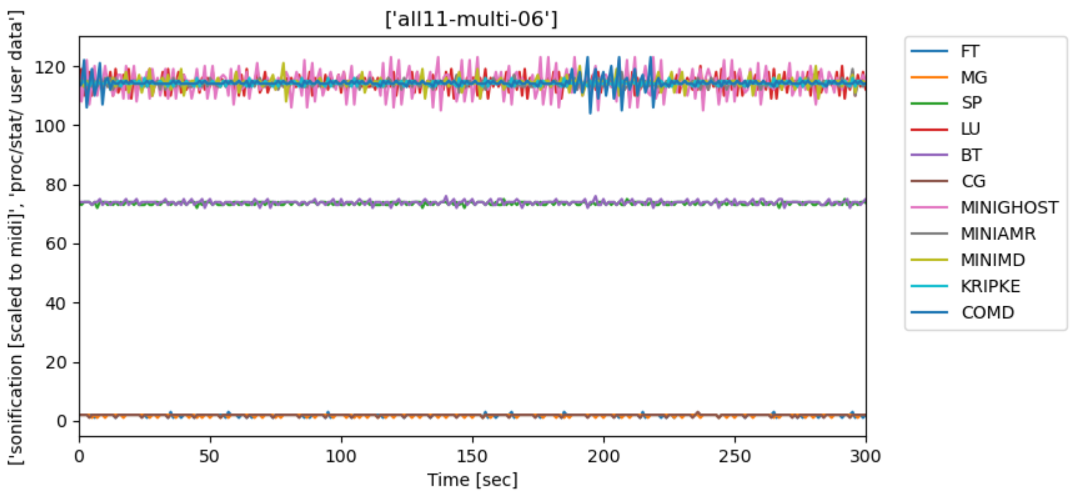
Experiment parameter:
App [multitrack over all apps], problem size [Z], run id [0], node id [average over all four nodes]

Figure 4.3: kripke all four node data as multitrack.



Surprisingly, three clusters are audible. The applications in the cluster with a very high pitch are: ft, lu, miniGhost, miniAMR, miniMD and kripke. There are two applications with a medium pitch: bt and sp. On the low pitch end there are: mg, cg and CoMD. These results are also shown in Figure 4.4.

Figure 4.4: All 11 applications data as multitrack.



4.7 Overall findings

The single-track mode did perform very well and made it possible to categorise the applications to a certain degree. As demonstrated in the experiments, the difference and difference2 mode were not providing meaningful insights. Surprisingly, the multi-track mode outperformed all my expectations. It was possible to differentiate between nodes while the difference mode had failed. Therefore, I conclude that sonification can be a very good complementary data interpretation to the traditional visualisation technique.

Chapter 5

Discussion

In this bachelor thesis I investigated sonification based on the development of a custom solution. The results of the experiments show that sonification of HPC data can work, is able to provide new insight and offers the possibility for an easier understanding of the data.

5.1 Strengths and weaknesses

While conducting the experiments, strengths and weaknesses of the Python software solution were exposed.

In a few cases it was possible to classify an application solely based on the sound produced (e.g. the CoMD application with its incredibly unique pattern of anomalies). On the contrary, there were a bunch of applications which were too similar to each other to distinguish them based on hearing the tracks.

Both modes difference and difference2 showed some flaws in mapping data to pitches. These modes were always able to show a huge difference independently of the scale of the actual size of the difference. The developed algorithm skewed the scale of the data so that the scope was always zoomed in to detect a difference. This made both modes unusable to show meaningful insights. To fix those modes it would be necessary to use the scale of the whole data set and to adjust the pitch scale accordingly (e.g. fixate the common pitch C^4 to the average and play deviations correspondingly higher or lower in pitch).

In contrast, the multi-track mode, which was only a showcase, surprised with a possibility to gain insight about parts playing out of tune (see Figure 4.2). This mode was also able to generate three clusters out of the 11 applications which showed an unexpected use case of this mode (see Figure 4.4). This mode could also be used in general to find similarities and dissimilarities.

The usability of the program could be improved in some ways. The configuration file could be improved with options for all runs, which is currently not implemented. This would look similar to the node option which is able to select node individually and has an option to select all together. As of right now, the data selection only works for the data from Ates et al. (2018) [2], a general approach or described API (e.g. the csv format or a custom "sonify" format) would be beneficial for further use.

5.2 Limitations

Some results are dependent on subjective perception. Somebody might not be able to hear very high pitches while others might not be able to differentiate between pitches due to lack of a musical ear. To remove the subjective perception a study with a subject group should be conducted. After listening to the tracks, the subject group could help to answer if the question of a classification is even possible solely based on the /proc/stat/ user data.

The software currently does not have any functionalities based on music theory. An example would be generating tracks which are using the general major scales. This could make the track much more enjoyable and distinguishable.

The possibility to gain new insights solely based on hearing as compared to the visualisation seems very limited, however it can be easier to detect insights by hearing rather than comparing multiple graphs.

5.3 Conclusion

The results show that this software can be used to compare applications, runs, different problem sizes and find differences between multiple applications and nodes, to compute the sound average over nodes, to map data to pitches and to generate sound with different instruments. However, there are still difficulties to overcome concerning the difference and difference2 modes.

Nevertheless, differences could be detected by the multi-track mode, which is based on another method.

More generally, this work shows that sonification of HPC performance data is successfully working and through sonification insights can be gained.

Chapter 6

Future Work

As already mentioned, if a study would show that we can distinguish applications solely on sound, this would open a new data analysis path for an application classification. This would open the possibility to use general audio analytics on HPC performance data as a new norm. It may also be possible to create a detector/classifier which would, only through listening to the application sound, be able to identify certain events and states (e.g. performance drops, idle times, load imbalance).

6.1 Software extension

The software could be extended in different aspects:

- Adding additional modes for different objectives. For example, a mode that searches for repeating patterns in different applications, which then could lead to new insights about the source code of the application. Another example would be to use different metrics or a combination of metrics at the same time.
- Extend the data selection to work independently of data sets.
- Add functionalities based on music theory.
- Expose more of the underlaying MIDI format to be able to fully manipulate the resulting tracks.

- Add a music player to the application, so that everyone that uses this application hears the same sounds. This is currently not assured because the MIDI format is interpreted by the users music player (the exception here is if FluidSynth was used to synthesis the MIDI to a Waveform).
- A configuration file generator could be added to reduce the effort for larger experiments.

6.2 Additional experiments

In the current state the software would also allow for many different experiments. For example, all 721 metrics from the data set of Ates et al. (2018) [2] are available. Other experiments could compare applications in depth against each other or focus on different parameters simultaneously.

Bibliography

[1] Wikipedia contributors. Sonification. https://en.wikipedia.org/wiki/Sonification, 2020. [Online; accessed 9-Feburary 2021].

[2] Emre Ates, Ozan Tuncer, Ata Turk, Vitus J. Leung, Jim Brandt, Manuel Egele, and Ayse K. Coskun. Taxonomist: Application detection through rich monitoring data. https://doi.org/10.1007/978-3-319-96983-1_7, 2018.

[3] Andrea Fasolo Rao. New media for scientific data visualization. https://issuu.com/andreaefferao/docs/newmediaforscientificvisualization, 2016. [Online; accessed 12-December 2020].

[4] Wanda L. Diaz-Merced, Robert M. Candey, Nancy Brickhouse, Matthew Schneps, John C. Mannone, Stephen Brewster, and Katrien Kolenberg. Sonification of astronomical data. https://doi.org/10.1017/S1743921312000440, 2012.

[5] Robert L. Alexander, Sile O'Modhrain, D. Aaron Roberts, Jason A. Gilbert, and Thomas H. Zurbuchen. The bird's ear view of space physics: Audification as a tool for the spectral analysis of time series data. https://doi.org/10.1002/2014JA020025, 2014.

[6] Datavized Technologies, Inc. Twotone. https://twotone.io/, 2019. [Online: accessed 11-December 2020].

[7] Erin Braswell. sonify. https://github.com/erinspace/sonify, 2017. [Online; accessed 14-December 2020].

[8] Ole Martin Bjørndalen and Rapolas Binkys. Mido. `https://mido.readthedocs.io/en/latest/`, 2020. [Online; accessed 14-December 2020].

[9] Wikipedia contributors. Midi. `https://en.wikipedia.org/wiki/MIDI`, 2021. [Online; accessed 11-Feburary 2021].

[10] Python Software Foundation. Supported ini file structure. `https://docs.python.org/3/library/configparser.html#supported-ini-file-structure`, 2021. [Online; accessed 26-January 2021].

[11] Tom Moebert. Fluidsynth. `https://github.com/FluidSynth/fluidsynth`, 2021. [Online; accessed 19-December 2020].

University
of Basel

Faculty of Science

Declaration on Scientific Integrity
(including a Declaration on Plagiarism and Fraud)

Bachelor's Thesis

Title of Thesis *(Please print in capital letters)*:
The Sound of Computing

First Name, Surname: *(Please print in capital letters)*	Maarten Schenk
Matriculation No.:	10-066-652

With my signature I declare that this submission is my own work and that I have fully acknowledged the assistance received in completing this work and that it contains no material that has not been formally acknowledged.

I have mentioned all source materials used and have cited these in accordance with recognised scientific rules.

In addition to this declaration, I am submitting a separate agreement regarding the publication of or public access to this work.

☐ Yes [✓ No]

Place, Date:	Olten, 15.02.2021
Signature:	[illegible]

Please enclose a completed and signed copy of this declaration in your Bachelor's or Master's thesis .