PARALLEL COMPUTING OF PARTICLE TRAJECTORY SONIFICATION TO ENABLE REAL-TIME INTERACTIVITY

Jiajun Yang

Ambient Intelligence Group
CITEC, Bielefeld University
Bielefeld, Germany
jyang@techfak.uni-bielefeld.de

Thomas Hermann

Ambient Intelligence Group
CITEC, Bielefeld University
Bielefeld, Germany
thermann@techfak.uni-bielefeld.de

ABSTRACT

In this paper, we revisit, explore and extend the Particle Trajectory Sonification (PTS) model, which supports cluster analysis of high-dimensional data by probing a model space with virtual particles which are 'gravitationally' attracted to a mode of the dataset's potential function. The particles' kinetic energy progression of as function of time adds directly to a signal which constitutes the sonification. The exponential increase in computation power since its conception in 1999 enables now for the first time to investigate real-time interactivity in such complex interweaved dynamic sonification models. We speeded up the computation of the PTS model with (i) data optimization via vector quantization, and (ii) parallel computing via OpenCL. We investigated the performance of sonifying high-dimensional complex data under different approaches. The results show a substantial increase in speed when applying vector quantization and parallelism with CPU. GPU parallelism provided a substantial speedup for very large number of particles comparing to using CPU but did not show enough benefit for a low number of particles due to copying overhead. A hybrid OpenCL implementation is presented to maximize the benefits of both worlds.

1. INTRODUCTION

Model-Based Sonification (MBS) is a sonification technique that involves (usually high-dimensional) data into the definition of dynamic systems which behave according to given laws of motion. Interaction modes (e.g. shaking, knocking) provide the means for interactively exploring features of the model (and thus coherences within the underlying data) as they result in a dynamic system behavior that directly contributes to a sound signal, i.e. the sonification [1].

Let's take an example of studying how a new percussive instrument works and sounds like. We can visually observe the construction of the object, then touch over the instrument and feel the materials of different parts. Then we can give it excitations through various physical actions, e.g. tapping, hitting, scratching, to test how each part of the instrument sounds like. To understand the sound it can provide, these excitation will also accompany with different dynamics, e.g. tapping it gently or hitting it hard. Furthermore, we can pick up other objects such as a drumstick to interact with the instrument to find out more acoustic possibilities. Through these, one can gain a good understanding of the functionality of the instrument.

Model-Based Sonification applies the similar analogy to the sonification for data mining. For instance, a dataset has its own intrinsic features which are unknown. However, we can define a dynamic model to bridge between the abstract data and the domain of acoustical systems, which are nothing but physical systems which react dynamically. Assuming that a model designer also implemented modes of interactions, the user can then also provide excitations to the data-driven configuration in model space and in turn the sonification model generates the acoustic response as immediate feedback to the user, whatever the unforeseeable interaction will be. The data imprints acoustic model properties which holistically and characteristically manifest in sound properties.

A Model-Based Sonification design usually consists of the following steps:

1. The *setup phase* defines how the data configure elements of the model (which typically exist in a model space). The data themselves are neutral and abstract, merely numbers. The setup grants them physical properties, to establish a *dynamic* system with internal degrees of freedom. For example, each data point can be considered a point mass in a d-dimensional space. Or each data point might be considered as a mass-spring system with inherent attributes such as mass, stiffness, thus an element that can oscillate and exhibit acoustic behaviors.
2. The *model dynamic* sets the equations of motion that ultimately determines the temporal evolution of the system and thus the sound. It can make use of physical principles such as inertia, friction, propagation, and energy conservation.
3. The *excitation phase* is the core part of the interaction to trigger auditory display. In this phase, specific excitation methods need to be defined, whether it is through simple Mouse clicks or complex tangible/physical interactions. This phase is also closely tied with the *model dynamic*.
4. The *link variables* are model-specific features that can be calculated at any time and whose temporal progression usually delivers directly the sound signal.

In data mining, *Exploratory Data Analysis* defines the process to acquire understanding of data and detect patterns when an explicit knowledge of the data is absent. In the 1970s, Tukey established many famous visualizations of uni- and multivariate data,

This work is licensed under Creative Commons Attribution Non Commercial 4.0 International License. The full terms of the License are available at http://creativecommons.org/licenses/by-nc/4.0

e.g. leaf plots, histogram. While they are still the predominant methods in data visualization, visualizing high-dimensional data is not always an easy job. For example, parallel coordinate [2] plots are commonly used to visualising multivariate data by juxtapositioning each data dimension along one axis. However, important patterns and features can be difficult to distinguish when the data space is dense [3]. Scatterplot matrices are particular useful when exploring the correlation between any two variables in the data space, but they lack the ability to examine the holistic feature of the data. The reason for bringing up the above examples is to show that there is a limitation with the visualization which we believe auditory display can help to overcome.

In the domain of sonification research, the techniques used in sonifying data have been predominantly circled around Parameter Mapping Sonification, Audification and Auditory Icon/Earcon-based displays. The former technique uses data to drive parameters of a sound generation engine (commonly a synthesizer). While there are many possibilities to create fruitful mappings from data to synthesis parameters, we cannot avoid the sonification researcher's own subjective decision on the mapping and scaling process. In Icon-based displays the two main techniques are Auditory Icons and Earcons, where collective sound pieces or melodies are used to represent certain state of the data vector. These types of auditory displays rely solely on a subjective mapping process. The design bias can be problematic in exploratory data analysis because of the absence of explicit knowledge of the data, e.g. switching the scaling or polarity of the mapping between a data channel and the pitch of synthesizer can lead to very different auditory patterns, prompting to different judgements.

2. LITERATURE OF SONIFICATION FOR EXPLORATORY DATA ANALYSIS

There is a body of work on parameter-mapping sonification for general data inspection[1]. For reasons of limited space we merely mention a few. An early on research of using auditory display in exploratory data analysis can be found in [4], where Flowers and Hauer looked into the efficiency of using auditory display to study data in comparison to histogram and Box-Whisker plots. Peres and Lane also studied the effectiveness of sonification of box plots in [5]. Flowers et al. examined data using auditory and visual display of scatterplots for bivariate data in [6]. The remainder of this section provides pointers to examples of developing general structure-specific sonification models for data analysis rather than targeting a specific type of dataset.

Hermann and Ritter introduced the Model-Based Sonifcation in [7], a paper in which they exemplify the idea with two sonification models: Particle Trajectory Sonification and Data Sonograms. In [8] they introduced a model to sonify data w.r.t. its principal curve. Further sonification models were introcuced in series, such as a local heat exploration model (LHEM) [9], Growing Neural Gas Sonification for exploring intrinsic dimensionality of data and the Markov-chain Monte Carlo Sonification model [10], most of them summarized in [11]. Multi-touch interaction for data sonograms using MBS was carried out in [12]. Crystallization Sonification was introduced in 2002 in [13], this model aims at exploring the intrinsic data dimensionality around user-selected points in relation to the global data dimensionality [14].

[1] throughout the ICAD proceedings

3. TOWARDS REAL-TIME INTERACTION IN PTS

The Particle Trajectory Sonification (PTS) aims at exploring the clustering of vectorial data [15]. This model was initially presented in [7] in 1999. At that time, however, the limitation in computing power did not allow a responsive (real-time) interaction to really treat the vectorial data as a 'virtual physical instrument', onto which the user can apply excitatory interactions to receive auditory feedback instantly in a tightly closed loop. Instead, it took up to minutes for reasonably complex problems to render usable sonifications, interrupting the control loop significantly. In 2016, the computing power has vastly improved. So for the first time, we can achieve (and explore the benefits of) a real-time experience of interaction with such complex sonification models in unprecedented detail and complexity.

The core objective of this research is to optimize the model to speed up in order to sonify multiple particles of larger high-dimensional datasets at a latency that is suitable for real-time interaction. To define a latency threshold to be able to consider as real-time interaction, also take into account that the interactive sonification is a type of sonic interaction. It is natural to assume a similar standard as the latency of a digital musical instrument, especially physical model instruments. In 2002, Wessel and Wright suggested that less than 10ms should be an ideal latency for performing a digital musical instrument [16]. A physical piano however can have even up to 35ms for pp notes [17]. Although it will be an ideal scenario if the MBS can be interacted within such a low latency level, it still seen rather ambitious when sonifying larger datasets. As mentioned in the later section (Sec 5), sonifying large and high-dimensional data may take upto a few seconds. Thus a significant speed up approach is required and also we loosen the definition of real-time interaction to within 300 ms in this particular case as user only needs to receive a relatively quick response for the purpose of analysis of the data rather than performing acute isochronous sequences as would be equipped for musical performance.

We start the presentation in Section 4 with details about the PTS and how this model can be dynamically sonified. Section 5 explores different approaches of implementing the sonification model, followed by a discussion and summary.

4. PARTICLE TRAJECTORY SONIFICATION

4.1. Model Definition

The Particle Trajectory Sonification Model is a Model-Based Sonification to analyze the clustering information of high-dimensional datasets through probing a data-driven potential function by dynamic test particles. Note that this model allows the analysis of multivariate data clustering without requiring to carry out any other clustering analysis beforehand.

The workflow of the model is as follows: a potential function $V(\vec{x})$ is constructed from the given high-dimensional dataset by superimposing data point potential functions which are centered in an Euclidean model space of same dimensionality as the data at each data point's coordinates. The overall potential is just the superimposition of all data point contributions at any arbitrary location in the high-dimensional model space. For sonification, particles with a given initial kinetic energy are injected into the model space. They move around according to a given dynamics (Newton's law plus a friction force). The resulting sonification

is obtained by adding the particles' kinetic energy as function of time.

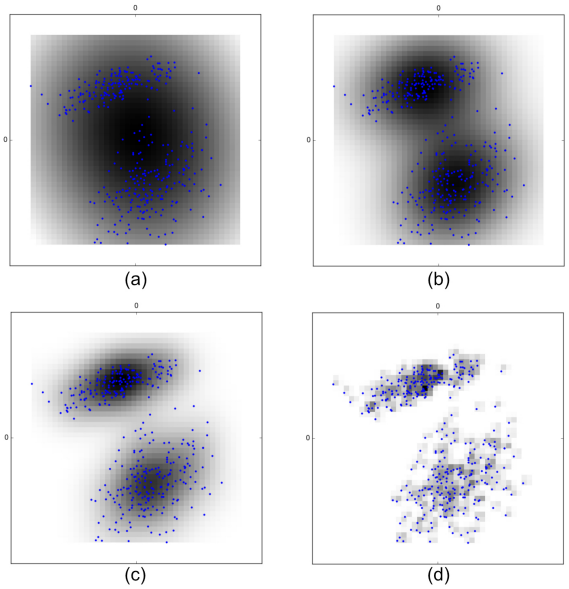

Figure 1: *Plots of the data potential for a 2D toy problem. (a)* $\sigma = 0.4$*; (b)* $\sigma = 0.16$*; (c)* $\sigma = 0.07$*; (d)* $\sigma = 0.01$.

For a formal definition, assume a given data matrix $X \in M(N \times d, \mathbb{R})$ whose row vectors are $\vec{x}_i^T, i = 1, ..., N$. The model space is an Euclidean vector space $\mathbb{R}^d$ in which points are fixed for each data point at coordinates $\vec{x}_i$. Assume an injected particle to have the coordinate $\vec{x}$, then the particle experiences a 'gravitational'[2] potential:

$$V(\vec{x}) = \sum_{i=1}^{N} \Phi(|\vec{x} - \vec{x_i}|) \tag{1}$$

where $\Phi(r)$ is the potential function of a data point defined by:

$$\Phi(r) = -\exp\left(-\frac{r^2}{2\sigma^2}\right) \tag{2}$$

where σ is the interaction length, which determines the resolution of the potential. Fig. 1 shows how σ affects the data potential $V(\vec{x})$ of a two-dimensional data set. When σ^2 is much larger than the average variation over dimensions, V exhibits only a single global minimum near the dataset mean (cf. Fig. 1a). With decreasing σ, local minima may arise corresponding to clusters in the data (cf. Fig. 1b & c). Yet if σ decreases further to smaller values than the average distance between the nearest neighbors of data points, each data points' potential trough is separated, thus we get N local minima (cf. Fig. 1d).

An injected particle of mass m is given a random initial kinetic energy W_{kin} yet so low that the particle can't escape from the data, i.e. $W_{kin} < -V(\vec{x})$. From $W_{kin} = mv^2/2$ we obtain the absolute velocity and set up its random initial velocity vector $\vec{v}$ in d dimensions. Newton's law of motions describe how the particle moves in model space. Numerical integration of the equation of motion with $\Delta t = d_t$ yields the following updates for the particle's position $\vec{x}$ and velocity $\vec{v}$:

$$\vec{v} := r\vec{v} + d_t \frac{-\nabla V}{m} \tag{3}$$

where r is the energy loss ratio due to friction of the model space.

$$\vec{x} := \vec{x} + d_t \vec{v} \tag{4}$$

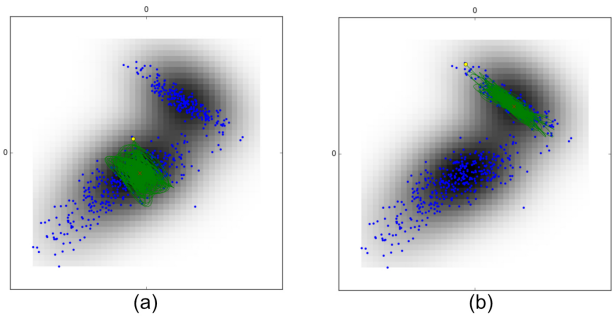

Figure 2: *Examples of the trajectories of two particles are injected in each one of the clusters in* 5000 *iterations. The green lines represent the trajectory. The yellow circle indicates the start coordinates of the particle and the red cross is the end of the trajectory.*

As aforementioned the sonification is the time series of kinetic energy[3]. The frequency and spectral complexity of the sound depend on rate of change of the velocity v. This means two key properties of the sonification in order for user to listen and understand the data:

- Particles attracted to a larger cluster exhibit a higher pitch than those attracted to a small cluster due to the stronger gravitational pull.
- If a particle started at the edge of a cluster, as it oscillates through the mode and (by friction) converges slowly to the cluster center (i.e. the mode), the spectral complexity of the sound decreases until eventually a sine wave like tone. This is due to the particle becoming less and less affected by the non-harmonic shape given by the tails of the potential function V .

An example of the particle movement can be seen in Fig. 2. Two particles were injected into different clusters (the yellow dots are the initial locations). After the initialization, the particles started moving in model space. They are pulled by the collective attractive forces of all model elements and due to friction they converge (in this case) to the local minimums of the clusters the particle belonged. Subject to different sizes of the clusters, their potential functions are different causing different velocity changes throughout the trajectories.

From the perspective of the time series of the particle's kinetic energy, the sonification is structurally merely the audification[4] [1] of the kinetic energy as a function of time.

[2] gravity works different in this model space than in our universe

[3] Equivalent to the square velocity vector since mass is a constant

[4] Audification is a technique that considers the data vector directly as an audio vector and plays the vector back in audio rate with downsampling or upsampling if required.

A video example of the sonification can be found in Sec. 7. Please refer to 'PTS-parameters.mp4' for a demonstration how σ and r affect the sound.

4.2. Exploring example data using the system

In Fig. 3, an example is presented of using the system to distinguish two separate clusters. In this example, we used a mixture of Gaussians to create data with the controlled features, here 4 clusters, $N = 980, d = 6$. The figure suggests that the right part of the scatter plot is a single cluster. However, the trajectories show different convergence targets. Fig. 3a & 3b each demonstrates the trajectory of particles attracted by different clusters. The particles converged to two different areas, leading to two different textures in sound as also rendered visible in the spectrogram respectively. This indicates that there are indeed two clusters whose discrimination is visually impossible as they overlap in the scatter plot.

Please refer to video file 'PTS-clusters.mp4' for a demonstration on how to use the sonification model for detecting different clusters, for finding cluster's edges/center and outliers.

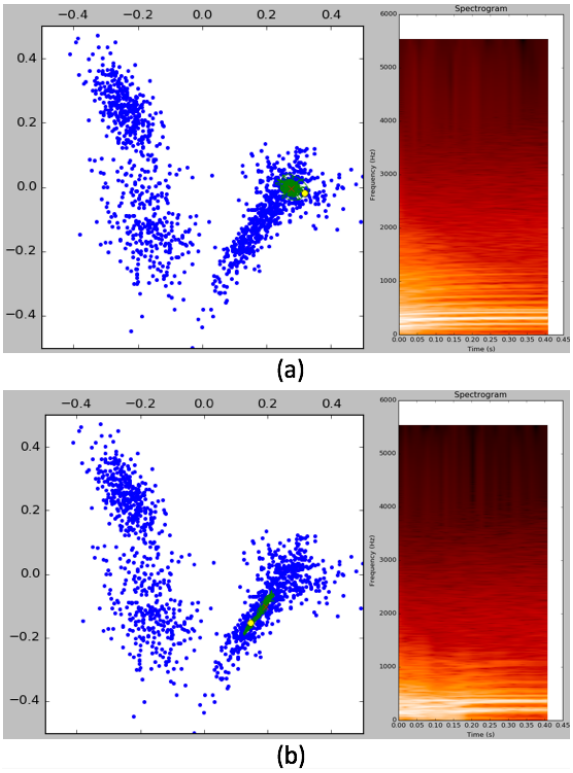

Figure 3: *Comparison of the two particle trajectories and their spectrograms.*

5. MODEL OPTIMIZATIONS WITH VECTOR QUANTIZATION AND PARALLEL COMPUTING

The previous section presented the mathematical model and sound examples for injecting a single particle. The analysis of clustering is about understanding the grouping of data points, which is a zonal property. Thus it will be more meaningful to inject multiple particles simultaneously and listen to their collective sound in a particular zone. The result sonification can either be listened individually or holistically by superimposing all particles' energy and then performing a normalization.

The following section presents multiple approaches of optimizing the dataset with vector quantization, as well as parallelism with OpenCL framework [18].

5.1. Baseline method

The baseline method is written is Python using the C-extension library Cython [19] for calculating the particle trajectory. The squared velocity vector is played back as sound using pyo's FIFO player [20] module. Using the FIFO player, the particle trajectory can be rendered in smaller chunks (per buffer size N_s) continuously that are then pushed to the queue of the FIFO player for continuous playback. A smaller buffer size can lower the latency as long as the rendering time t is below $t < N_s/f_s$, where f_s is the sampling rate. For PTS, we choose a sampling rate of $f_s = 11025$ because there is very little audible information in the sonification at higher frequencies. We use a buffer size $N_s = 256$. This implementation is originally targeted for single PTS.

In terms of the trigger latency, we define 300 ms as the threshold below which interaction (i.e. insert particles and hear back the sound) can be regarded as real-time.

As for the performance, the computational cost is proportional to the number of particles, which was set between 1 to 100. As shown in Fig. 4, for smaller datasets (200×5), Cython implementation can still achieve satisfactory latency for real-time interaction. As for the 2000×5 dataset, rendering multiple-particle trajectories slowed down significantly, and it became unsuitable above 30 particles. This results prompts to a requirement for significant speedup.

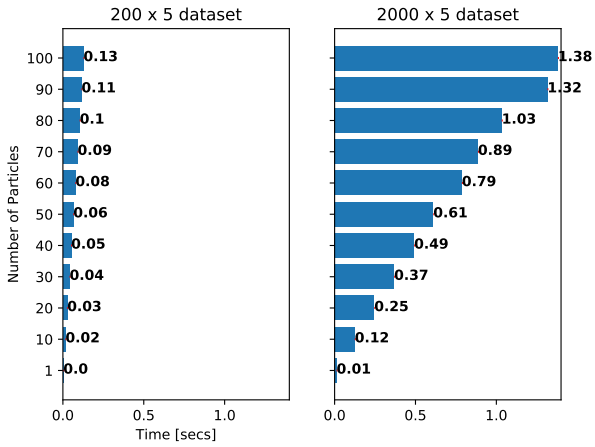

Figure 4: *Performance for different numbers of particles using Cython without parallelization.*

5.2. Vector Quantization

When dealing with larger datasets, e.g. $N > 10^3$ data points, vector quantization allows to reduce the resolution while still maintaining the general clustering structure. Here, we apply the k-means vector quantization to reduce the size of a given dataset into k prototypes minimizing the cost function

$$E = \sum_{i=1}^{N} \sum_{j=1}^{k} h_{ij} \left\| \vec{x}_i - \vec{v}_j \right\|^2 \quad (5)$$

where $h_{ij} = 1$ if $\vec{x}_i$ is nearest to prototype $\vec{v}_j$, else $h_{ij} = 0$. Note that the number of prototypes is still as high as possible, e.g. around 1000, and not selected according to the number of expected clusters. A detailed analysis as to how data set reduction affects the PTS will be published elsewhere. Fig. 5 shows a linear change (note that both x and y-axis are log-scaled) in the computational time of the PTS algorithm with 1 particle only after applying different levels of vector quantizations to a $10^4 \times 5$ dataset. Based on

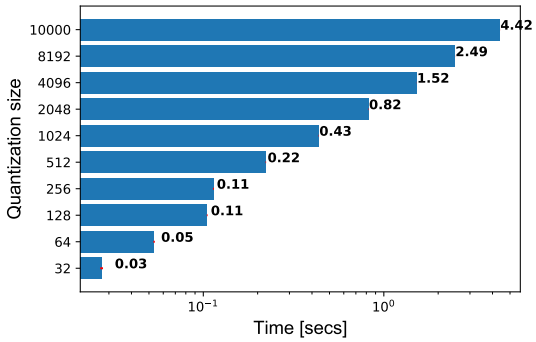

Figure 5: *Benchmark of the vector quantization of a $10^4 \times 5$ normalized dataset. The test was run on a Windows desktop with Intel i5-3350 2.7GHz CPU.*

Fig. 5, in order to have a responsive interaction for a single particle (i.e. latency < 0.3 s), the total numbers $N \times d$ of a data matrix should be approximately < 15000. Thus, given a new data matrix $X \in N \times d, \mathbb{R})$, if the total number of cells is greater than the threshold, a mild vector quantization is of k prototypes is applied, e.g. with $k = \text{round}(15000/d)$.

The vector quantization can be precompiled prior to interaction and sonification as a data reduction method, thus the compute time for the quantization does not affect the time for the sonification.

5.3. Parallel Computing with OpenCL

We paralleled the computation of multiple particles' sonification using OpenCL. The OpenCL framework allows parallelization to be executed on either CPUs or GPUs. In both cases, the N_p particles are distributed to the parallel kernels. Each kernel shares the same data matrix and model parameters, the only difference is the initial location of the particle. As a result, each kernel returns the trajectory and kinetic energy vector for the assigned particle. The performance results are presented in the next subsection.

5.4. Performance of Different Process Units

We tested the new model implementation on 1 CPU (Intel(R) Core(TM) i5-5287U CPU @ 2.90 GHz, dual- core) and 2 GPU Nvidia GeForce 940M and Nvidia GeForce GTX 970). These three units cover a common range of consumer processing units. Due to the limitation in equipment, the latest generation graphic cards in 2017 such as GeForce GTX 1000 series are not tested. However, our selection shall provide an insight into the performance of the PTS model using common computers.

Two random datasets are selected for the benchmarks. A smaller data matrix $X_1 \in M(200 \times 5, \mathbb{R})$ and a larger data matrix $X_2 \in M(2000 \times 5, \mathbb{R})$. Notice that the total number of cells of X_2 equals the threshold for applying vector quantization, hence vector quantization was not applied.

Comparing the baseline result from Fig. 4 with using OpenCL with CPU (Fig. 6), the latter achieved a mean acceleration ratio of 1.67 in X_1 and 1.59 in X_2 for multiple particles ($N_p > 1$).

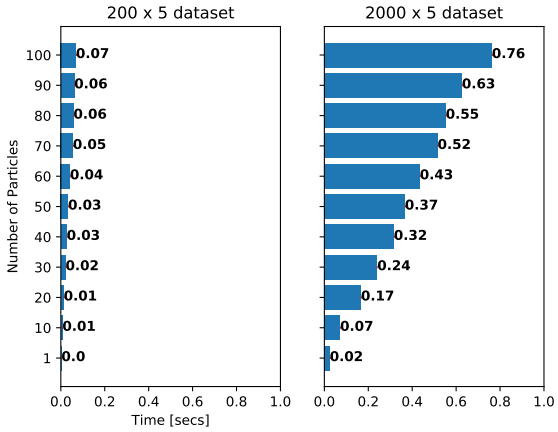

Figure 6: *Performance of PTS written in OpenCL using the Intel(R) Core(TM) i5-5287U CPU @ 2.90GHz.*

When testing with the GPUs, Fig. 7 & 8 show that there is a constant offset in the computation time, which is due to the data being copied in and out of OpenCL kernel at each call. The offset time is proportional to the dataset size, thus for the larger dataset X_2 the latency is over threshold. On the other hand, since the OpenCL implementation paralleled each particle, increasing the number of particles does not lead to a proportional increase in the computing time. Further tests with a higher number of particles $N_p > 10^3$ showed that compared to CPU we can gain a 30-fold speedup before compute time starts to increase linearly also with GPU.

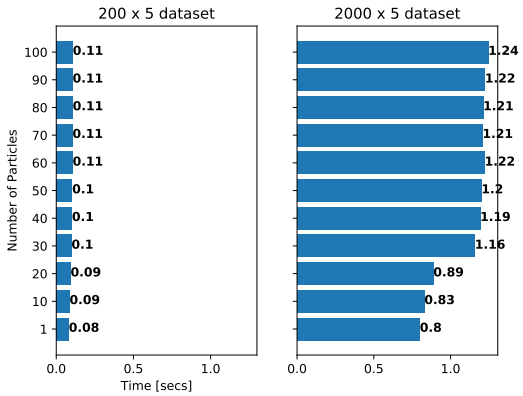

Figure 7: *Performance of PTS written in OpenCL using the Nividia GeForce GTX 970 graphic card.*

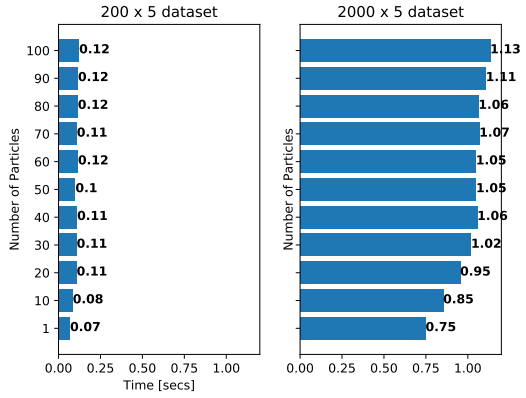

Figure 8: *Performance of PTS written in OpenCL using the Nividia GeForce 940m graphic card.*

5.5. Hybrid parallelism between CPU and GPU

Section 5.1 – 5.4 led to the following conclusions:

1. The Cython implementation is 'in serial' thus the computation time increased linearly as N_p increased.
2. Comparing this to using Cython and OpenCL on Intel i5 CPU, the OpenCL implementation is faster. But it is not efficient at dealing with larger amount of particles.
3. Implementing OpenCL on GPU has a strong ability for paralleling large amount of particles but currently prolonged by the data I/O overhead (offset) between memory and graphic card units.

Based on these 3 results, we propose a hybrid OpenCL solution that utilizes the CPU for smaller N_p but then automatically switches to GPU-based rendition if N_p becomes larger. The variation in N_p will depend on the user's interactions, e.g. tapping an area with finger or palm. The threshold may vary due to the computer's processing power, yet it can be pre-calculated when a new dataset is loaded.

We conducted a test to study how the following three key variables that affect the computational time:

- $N_D = N \times d$ is the total size of the dataset. This variable affects is the main factor for the data input overhead when using GPU. It also contributes the cost of the potential function (cf. eq. (1)).
- N_v is the trajectory vector (aka. the audio buffer size), which affects the overheads of both input and output (passing the vector from GPU back to memory).
- N_p is the number of particles, which only affects the trajectory's computation cost.

From our benchmarks we can model the relationships between the three aforementioned variables and computational time. For GPU we expect:

$$t = aN_DN_v + bN_v + \begin{cases} cN_DN_v & \text{if } N_p <= N_p^o \\ c\frac{N_p}{N_p^o}N_DN_v & \text{else} \end{cases} \quad (6)$$

whereas for CPU, we expect

$$t = dN_DN_vN_p \ , \quad (7)$$

where a, b, c, d are coefficients and N_p^o is the threshold of maximum GPU capacity for calculation in parallel at at time. Eq. (6) addresses that the computation cost is based on three parts: input offset, output offset and cost for rendering the sonification and trajectory. However, the cost does not increase when $N_p \leq N_p^o$. Using OpenCL with CPU does not suffer from the copying offsets but only lacks the benefit of allowing a larger number of particles to run without affecting the time.

We then tested the assumption using the GTX970 graphic card and the Intel i5 CPU with different N_D, N_v and N_p and their correspondent computational time t (cf. Fig. 9). However, instead of plotting t as function of N_P, we depict $t/(N_D \cdot N_v)$, which gives the unit computational time that is only relevant to the number of particles plotted on the x-axis. In Fig. 9, each line (apart from the black straight line) represents a specific combination of N_D and N_v. The black straight line is the linear regression of the CPU's OpenCL performance[5].

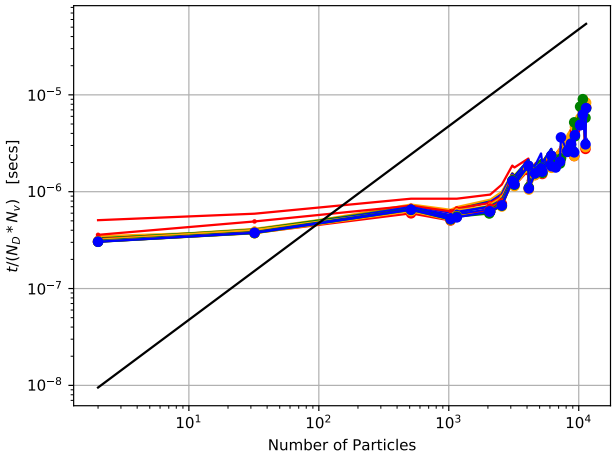

Figure 9: *Normalized computation time ($t/(N_DN_v)$) comparison. The markersizes of the data points represents is mapped to N_D. Color coding is mapped to N_v.*

The figure shows that the approximate maximum number of particles for which parallel computing most significantly increases performance (N_p^o is peaked at 2048, from then on the required time increased linearly with N_p. When $N_p < N_p^o$, the computation cost stayed relatively flat with minor increase as N_p increased, which is due to $N_p \times N_v$ of kinetic energy vector (for sonification) and $N_p \times N_v \times d$ of position matrix (for visualization) to be copied from GPU back to the memory. Also, the offset of the first data point ($N_p = 1$) indicates the input offset. At a larger number of particles ($N_p > 100$), GPU shows superiority over CPU and the speedup exhibits a maximum at $N_p^o \approx 2048$, which is about 30 times faster than CPU.

In result, for the tested hardware configuration we arrive at the following conclusions:

1. It is more effective to use OpenCL with CPU than GPU when $N_p < 100$, vice versa.

[5]The high linearity we found when using CPU at lower particle number (< 100) led us to use linear regression line rather than the actual computational cost.

2. When $100 < N_p < 2048$, it is more efficient to use GPU to better exploit the potential of parrallelism.
3. When $N_p > 2048$, the speedup is maximized with a factor of ≈ 30.
4. A significant overhead persists for our current approach due to data being copied in and out of the GPU.

We then put the hybrid method into test and set the switching threshold to 100. In this test (cf. Fig. 10), we tested different numbers of particles and data sizes. As mentioned before, 300 ms is the latency threshold in order to be considered suitable for real-time interaction. Based on the graph, most of the tested data sizes are suitable for real-time interaction when N_p is smaller than 100. For $N_D < 1400$, low latency can still be still achieved for higher N_p values. But as $N_D > 1400$ increased, the latency is greater than 300 ms for larger numbers of particles. However, this issue can be addressed by using the above-proposed vector quantization approach presented in Sec. 5.2.

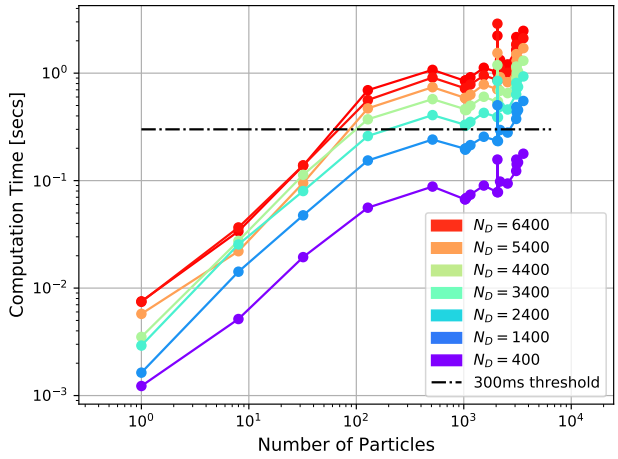

Figure 10: *Plot of the computation time t with the hybrid method against N_p. The dashed line indicates our defined threshold for real-time interactivity.*

6. DISCUSSION & CONCLUSION

This paper continues the research on Model-Based Sonification. For the example of Particle Trajectory Sonification we implemented data optimization and parallel computing procedures to increase the responsiveness when interacting with high-dimensional complex data. The PTS model was initially designed to analyze clustering features of usually high-dimensional datasets offline as the computational time may take up to minutes and even longer. In this project, we proposed several methods to increase the total performance when sonifying multiple particles' trajectories for larger datasets. This could pave the way towards an interactive structure-specific[6] toolbox for the exploratory analysis of high-dimensional data. Firstly, vector quantization can effectively set up a time cap of 300 ms for a single particle trajectory while maintaining the data structure and thus enable PTS for larger datasets. Secondly, we observed that the OpenCL implementation via CPU provides a moderate speedup compared to the baseline Cython implementation. With a higher amount of GPU process units, increasing the amount of particles does not increase the time proportionally. At our test, we found a maximum of 30 times faster computation as compared to CPU for larger number of particles (> 2000). However, the data I/O overhead between memory and the graphics card is still large for our current implementation. For the tested computer configuration, the hybrid OpenCL implementation can ensure low latency ($<$300 ms) for $N_p < 100$ even with large dataset, but it is found to be difficult to keep the latency low as the number of particles N_p becomes very large.

For our next steps, we plan to reimplement our OpenCL algorithm in hope to eliminate input offsets of GPU by allowing the data matrix to remain stored in the graphic units instead of passing it at each call. Also, we currently work on a distance-matrix-based method for choosing the most influential neighbors relative to the current particle's position at each iteration and discarding the rest. This could lead to another significant speed up in combination with vector quantization. These together could potentially lead to a significant speedup allowing thousands of particles to be sonified in high-dimensional datasets in real-time.

The advantage of listening to the clustering information via PTS over the visualization of the potential map (cf. Fig. 1, 2), is that potential maps can only be computed for rather low-dimensional problems (< 3 dimensions), whereas with few simple sound probes, the user can navigate the full potential function and quickly explore prevalent potential troughs (corresponding to clusters) at different scales of resolution.

[6] in contrast to data-specific

7. LINK

Supplementary material for this paper (media files) are available via the DOI: `10.4119/unibi/2911345`

8. ACKNOWLEDGMENT

This research was supported by the Cluster of Excellence Cognitive Interaction Technology 'CITEC' (EXC 277) at Bielefeld University, which is funded by the German Research Foundation (DFG). We wish to thank our student worker Akhil Jain for programming support.

9. REFERENCES

[1] T. Hermann, A. Hunt, and J. G. Neuhoff, Eds., *The Sonification Handbook*. Logos Verlag, 2011.

[2] A. Inselberg and B. Dimsdale, "Parallel coordinates: A tool for visualizing multidimensional geometry," *Proc. IEEE Visualization*, pp. 361–378, 1999.

[3] M. Graham and J. Kennedy, "Using curves to enhance parallel coordinate visualisations," *Information Visualization, 2003. IV 2003. Proceedings. Seventh International Conference on*, pp. 10–16, July 2003.

[4] J. H. Flowers and T. A. Hauer, ""sound" alternatives to visual graphics for exploratory data analysis," *Behavior Research Methods, Instruments, & Computers*, vol. 25, no. 2, pp. 242–249, 1993.

[5] S. C. Peres and D. M. Lane, "Sonifcation of statistical graphs," *Proceedings of the 2003 International Conference on Auditory Display*, 2003.

[6] J. H. Flowers, D. C. Buhman, and K. D. Turnage, "Cross-modal equivalence of visual and auditory scatterplots for exploring bivariate data samples," *Human Factors: The Journal of the Human Factors and Ergonomics Society*, vol. 39, pp. 341–351, 1997.

[7] T. Hermann and H. Ritter, "Listen to your data: Model-based sonification for data analysis," *Advances in intelligent computing and multimedia systems*, vol. 8, pp. 189–194, 1999.

[8] T. Hermann, P. Meinicke, and H. Ritter, "Principle curve sonification," 2000.

[9] T. Bovermann, T. Hermann, and R. Helge, "The local heat exploration model for interactive sonification," *International Conference on Auditory Display*, 2005.

[10] T. Hermann, M. H. Hansen, and H. Ritter, "Sonification of markov chain monte carlo simulations," *International Conference on Auditory Display*, 2001.

[11] T. Hermann, *The Sonifcation Handbook*. Logos Verlag, 2011, ch. Model-Based Sonification, pp. 399–425.

[12] R. Tünnermann and T. Hermann, "Multi-touch interactions for model-based sonifcation," *Proceedings of the 15th International Conference on Auditory Display*, 2009.

[13] T. Hermann and H. Ritter, "Crystallization sonifcation of high-dimensional datasets," *Proceedings of the 2002 International Conference on Auditory Display*, 2002.

[14] T. Hermann, "Sonification for exploratory data analysis," Ph.D. dissertation, Bielefeld University, 2002.

[15] T. Hermann and H. Ritter, "Model-based sonification revisited—authors' comments on hermann and ritter, icad 2002," *ACM Transactions on Applied Perception (TAP)*, vol. 2, no. 4, pp. 559–563, 2005.

[16] D. Wessel and M. Wright, "Problems and prospects for intimate musical control of computers," *Computer Music Journal*, vol. 26, no. 3, pp. 11–22, 2002.

[17] A. Anders and E. Jansson, "From touch to string vibrations - the iniitla course of the piano tone," *The Journal of the Acoustical Society of America*, vol. 81, no. S1, pp. S61–S61, 1987.

[18] J. E. Stone, D. Gohara, and G. Shi, "Opencl: A parallel programming standard for heterogeneous computing systems," *Computing in Science & Engineering*, vol. 12, no. 1-2, pp. 66–73, 2010.

[19] S. Behnel, R. Bradshaw, D. S. Sljebotn, and G. Ewing, "Cython: C-exsions for python," 2008. [Online]. Available: http://cython.org

[20] O. Bélanger, "Pyo," 2012. [Online]. Available: http://ajaxsoundstudio.com/team/