A CASE STUDY OF DISTRIBUTED MUSICAL AUDIO ANALYSIS USING THE GEDDEI PROCESSING FRAMEWORK

Gavin Wood
Department of Computer Science

Simon O'Keefe
University of York, York YO10 5DD, UK

ABSTRACT

Audio signal processing and refinement is an important part of a content-based music information retrieval system. As the our repertoire of techniques becomes more varied, there are greater requirements of computation power. Distributed storage techniques have become widespread and almost invisible with the advent of file-sharing systems, on-line digital music stores and on line storage services. Even discounting data with potential copyright entanglements, there is a vast amount that is ripe for analysis, and thus parallelised and distributed processing techniques seem increasingly appropriate.

Existing frameworks are already capable of a significant amount of audio analysis for music information retrieval. However they are by and large ignorant of distribution and parallelisation. There are middleware libraries to help with aspects of distributed computing, but combining the two can be cumbersome and inefficient.

This paper provides a brief description of a software framework that can process audio in a scalable and distributed manner: Geddei. The paper then takes an interesting and relevant signal analysis task often used for music information retrieval and implements it under the Geddei framework. The ease of use is discussed and various measurements taken of Geddei, both in comparison to itself under different circumstances, and 'reference code' that was used in a previous study. We discuss the problems with the distribution of the task with Geddei and offer some possible solutions.

1. INTRODUCTION

Almost all methods of content-based analysis of musical audio for information retrieval (IR) rely to some degree upon a signal (pre-)processing technique, and this technique can often be at the heart of the method itself. There exist several (open) systems for audio signal processing with respect to music IR, not least Tzanetakis's Marsyas library framework [1, 2].

With music IR and audio signal processing in general, we find that the amount of data to be processed is large. There is significant impetus to define our problems in more independent, declarative terms as distribution becomes more common in both software (e.g. threading) and hardware (e.g. grid technology). Distributive processing allows us to combine multiple individual CPUs (processing chips) to perform one task as a whole.

The problem of distribution may be phrased as one of organisation—we have a large task at hand, and we must break it down into smaller tasks that can be done *concurrently* and *independently*. With a large amount of data, the overhead of transport, and thus distribution, is large—most notably when compared to the processing tasks. This can be a problem when trying to find an optimal distribution method, since transporting data to another processing host makes little sense if the overhead of transport is comparable to the cost of processing. Thus the smaller tasks must be organised in such a way that maximises use of the resources available and minimises unnecessary transport.

1.1. Applications

As techniques become more complex, more varied and more adaptive, music IR methods require more computing power to (pre-)process audio signal data into forms that are useful for analysis techniques. We see ongoing efforts [3] to create a large and protected music database for benchmarking music IR techniques. A distributed generalised system could help in providing a common platform on which large scale processing benchmarks can be executed. As we see more large computing grids emerging, the music IR community needs to be able to use such technology effectively and easily. A simple and efficient distribution framework designed for music signal processing goes some way to solving this problem.

1.2. Related Work

Several toolkits already exist for the processing and analysis of audio signals. Libraries such as the Simple Utility Classes (SUCs)[4] provide basic programming components for signal-analysis, though distribution and parallelism in general is not addressed. Marsyas provides a broad programming interface for implementing many ideas found in music information retrieval, and addresses both traditional 'bottom-up' designs, as well as prediction-driven architectures. Its dataflow mechanisms are robust but potentially restrictive [1] . Again, no implicit parallelism is available natively.

[1] Marsyas supports processing atoms that are able to be given only a fixed-amount of data from one atom and provide some other fixed-amount of data to another atom

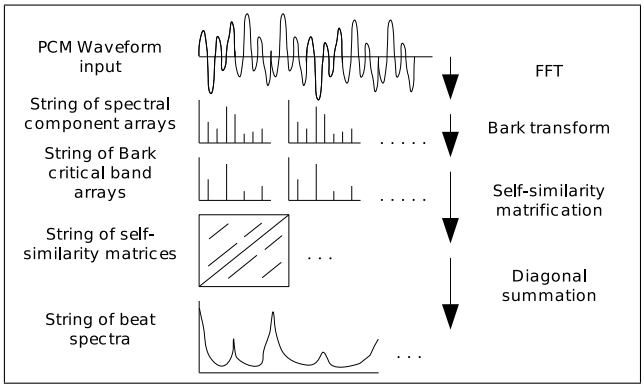

Figure 1. An activity flow chart of the beat spectrum analysis technique.

2. ANALYSIS TECHNIQUE

For the case study, an analysis technique had to be chosen that provided a useful datagram for music feature extraction, yet provided some method of testing against a real-world legacy version.

There exist several similar mechanisms for finding rhythmic information from musical data, be it either in symbolic form or as an audio signal. Tzanetakis, Essl and Cook [7] introduce the beat histogram, formed by autocorrelation of wavelet transforms. There is also the technique used by Rauber and Fruhwirth [8] which is essentially a FFT on selected Lagrange transform spectral bands. Each of these present data refined to describe the rhythmic properties of the incoming music. Tzanetakis et al. especially shows that the rhythm metric is an important tool for genre classification problems.

The beat spectrum reports a spectrum of frequencies indicating rhythmic similarity; peaks at a frequency suggest a strong rhythm at that frequency in the incoming audio. The discussed technique was used in Foote, Cooper and Nam [5] for measuring musical recognition though similarity. The technique is also used in Wood & O'Keefe [6] for measuring music similarity. Both papers show that the beat spectrum can provide a useful datagram for music similarity and analysis in general.

The beat spectrum was chosen as it fulfils both requirements well. As it was implemented for a previous study it gives a useful reference point for comparison.

2.1. Method

We take 30-second windows of the incoming audio data, each overlapped by 15 seconds. An array of short-time Fourier transforms is then calculated over this 30-second window using a window size of 2048-samples and a 50% overlap.

The standard beat-spectrum method was altered slightly to use a psychoacoustic transformation in the form of Bark critical banding [9]. This reduces a potentially large spectrum into a compact 24 'critical' band representation, based upon empirical studies of the ear and how we perceive pitch. Experiments from Wood & O'Keefe [6] suggest that this decreases computation time significantly with little or no effect on the fidelity of the results (from a music recognition point of view, at least).

The beat spectrum is formed from the summation of super-diagonals on a self-similarity matrix. The self-similarity matrix is formed by measuring the 'similarity' between two points in time of some incoming music. For an incoming audio signal, a useful similarity measure can be computed from the cosine distance between the two spectra which represent the frequency components of the music audio at those points in time; we use this measurement here.

3. GEDDEI'S ARCHITECTURE

Geddei is an acronym for General Environment for Distributed Dataflow Experimentation and Investigation. It provides a simple, transparent, declarative style interface for tasks that can be arranged as a data flow-orientated problem. It is highly scalable, equally suited for small-scale investigation and batch processing. It is highly efficient, using mechanisms such as cyclic shared buffers to maximise signal-data throughput. A plugin design allows simple addition of third-party processing modules. It is useful to consider the two primitives that are used when constructing data flow networks for audio analysis in Geddei:

Processor: Processor objects are the fundamental component of all Geddei processing modules. They represent an atomic and independent task of the whole problem, and as such represent the granularity of the distribution. Almost any single thing in a typical dataflow network can be likened to a processor object. This might include the FFT, the similarity matrification or a downsampler. Even audio file players are modelled as processors. This level of generalisation provides the basis for automation and distributability.

Connection: Connection objects provide a mechanism of data transfer between two Processor objects. Connection objects take care of getting signal data between processors regardless of their actual 'positions' in the system. A connection linking processors residing on different machines will be different to (and slower than) one linking processors residing in the same area of computer memory. However the level of abstraction means that the processors are unaware of this.

4. EVALUATION

With Geddei, distribution over two hosts was as simple as having all processing objects reside on one host. No specific optimisations needed to be added nor did the code need to be changed in any way for use on different architectures, leading to a very low prototyping time.

The reference code used was that developed for an earlier study [6][2]. It uses the Simple Utility Class library; no special optimisations were made at the time, however the simplicity allows the code to run with some degree of

[2] The code can be found at `www-users.cs.york.ac.uk/~gav/ref.tar.gz`

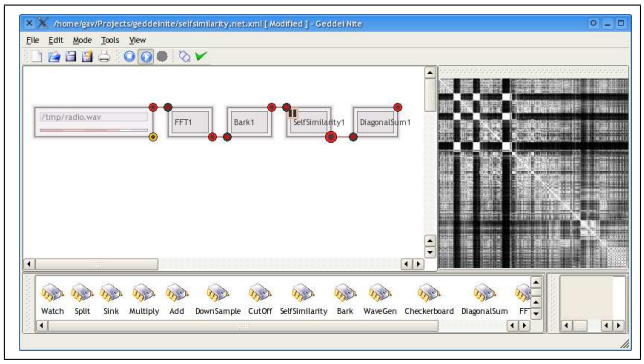

Figure 2. Geddei Nite, the Geddei network editor, with the beat spectrum analysis network, watching the output of the self-similarity matrix.

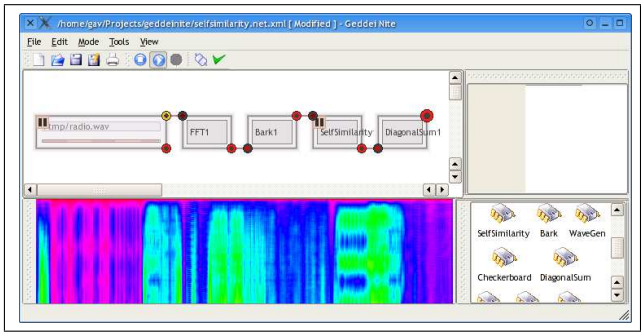

Figure 3. As figure 2, but changing the focus to the beat spectrum's output.

efficiency. Depending on the programmer's ability and the framework used, it may be likened to an average, sequentially-based implementation of some audio preprocessing.

4.1. User Interface

Networks are created and edited with Geddei-Nite. It is simple to inspect the data travelling through any channel in realtime without disrupting the actual data, much in the same way a wiretapper would evesdrop on a phone call unnoticed. This can be used to investigate some (hypothetical) unexpected or otherwise interesting result.

Geddei's strict and extensive signal-typing mechanism allows Geddei-Nite to probe the signal type and attach the relevant viewer, allowing an informative graph to be drawn. Figures 2 and 3 show the same network with differing views.

4.2. Performance

The performance evaluation of Geddei is split into 3 parts: We compare the Geddei and reference implementation on a single CPU workstation. We then test Geddei's distribution capacity by comparing non-distributed times to distributed. Finally we compare the effects of changing the amount of data to be processed between the Geddei and reference implementation. In all experiments, the length of the music track to be analysed was 279 seconds.

The FFT stage was given three variants: It was either left untouched, a low-pass filter was applied at 8KHz,

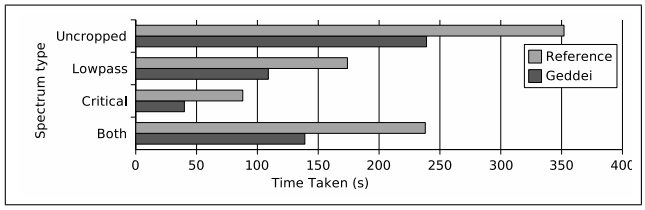

Figure 4. Comparison of the two systems' performances on an undistributed single CPU system. Specification: Athlon XP 2100; Linux 2.6.3; gcc 3.3.2.

or Bark-critical banding was applied. A further analysis was made that attained *both* the low-pass filter results and the Bark-based results. These change the dimensionality of the spectra significantly (between 1024, 186 and 24), moving the potential bottlenecks of the system to and from the self-similarity matrix.

Comparisons between the two implementations are relatively easy; Geddei has a built-in monitor processor that collects the output data and measures the time taken for processing. For the reference software, the GNU 'time' command was used to report the total time the CPU spent executing the program [3]. This could potentially be favourable to the reference software as the Geddei timing method measures real time taken rather than CPU time taken, however on an otherwise unused system they are similar enough for the purpose of this report.

This first test results are shown in figure 4. Geddei takes on average 38% less time on each task, a significant improvement. As would be expected, both Geddei- and reference-based implementations take less time to calculated both types of beat spectrum in one run than in two.

The scalability of Geddei was tested by comparing it to itself under while running under differing situations. Two hosts were used for this, as is described in figure 5. A restriction was placed on the analysis network that the music must start, and the analysed output end, on the workstation machine (host 1). Figure 5 shows the results. A theoretical maximum was calculated using the ideal parallelised-linear-resistance formula:

$$P_{combination} = (P_{host1}^{-1} + P_{host2}^{-1})^{-1} \qquad (1)$$

This formula is simplistic and gives only a basic idealistic marker. Problems such as different CPU architectures giving different performances depending on task and the software's running time only vaguely approximating as linear make this a rough guide at best; its inclusion here is to help visualise a potential limit of computation time, given that the two hosts are of differing speeds it is otherwise difficult to do.

Several distribution configurations were tried and the optimum selected. Those configurations are detailed below. Note that the superscript gives the host number they were distributed onto. They appear in the same order as they are shown in figure 5. The audio source and data reception processors were both on host 1, the designated workstation.

[3] this does not include any time the system spent executing OS tasks

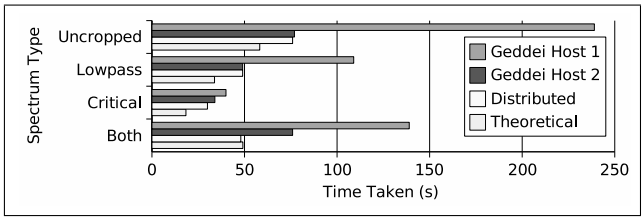

Figure 5. Comparison of Geddei's performance when distributed over two single CPU hosts. Specifications: Host 1: Athlon XP 2100; Linux 2.6.3; gcc 3.3.2. Host 2: Pentium 4 2.66GHz; Linux 2.4.19; gcc 2.2.4.

$$FFT^1 \rightarrow SSM^2 \rightarrow Diag.Sum.^2$$

$$FFT^1 \rightarrow BandPass^1 \rightarrow SSM^2 \rightarrow Diag.Sum.^2$$

$$FFT^1 \rightarrow Bark^1 \rightarrow SSM^2 \rightarrow Diag.Sum.^2$$

$$FFT^1 \rightarrow BandPass^1 \begin{array}{l} \nearrow \\ \searrow \end{array} \begin{array}{l} Bark^1 \rightarrow SSM^1 \rightarrow Diag.Sum.^1 \\ SSM^2 \rightarrow Diag.Sum.^2 \end{array}$$

From figure 5 we can see that host 1 is much slower than host 2. This is a particularly difficult problem to distribute; there are relatively few processing atoms and only the similarity matrix has a significant processing requirement, thus we can see Geddei's distributive capabilities being taxed.

The very-high-bandwidth dataflow restricts Geddei's ability to distribute the problem well between the two hosts, and for the three basic tasks, it is only a little faster than running on host 2 alone. The difficulties are overcome well when both the Bark-based beat spectrum and the low-pass-filter-based beat spectrum are calculated, attaining a slightly better time than the idealistic marker.

Finally the completion times were recorded for differing sizes of beat spectrum and low-pass filter cutoffs. Changing the low-pass filter cut-off alters the dimensionality of the vectors for calculating the cosine distances in the self-similarity matrix. It has the effect of reducing data (and thus computation time) at the cost of reducing the fidelity of the results (though most people agree that anything over around 4-8KHz is probably useless for the task of music IR).

Changing the matrix window size will change the scope of the beat spectrum; we keep the window hop distance fixed at 50%. Figure 6 shows the results of these tests.

Both graphs are linear confirming that Geddei is performing akin to the reference design. The lines of best fit suggest that Geddei is approximately twice as fast as the reference design, presumably heralding its parallelised layout, high-speed buffers and optimisation-friendly design.

5. CONCLUSION

We have introduced the Geddei framework and provided an example of its use for music information retrieval. We have shown that Geddei is well-suited to the design and development of music signal processing tasks and that it runs at least as fast, and often significantly faster than an—albeit simple—existing implementation. We have shown that it has some potential for distribution of music IR processing and can distribute well under good conditions, though there is clearly room for improvement in the worst-case scenario.

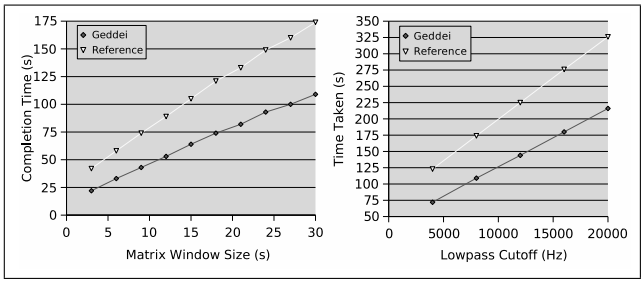

Figure 6. Comparison between the reference and Geddei implementations performance when either the matrix window size (left) or the low-pass filter cutoff (right) is changed. Both are on the same host as figure 4.

Immediate future work will involve increasing Geddei's repertoire of component techniques and extending its distribution capacity to help with the problems of distribution granularity. We also want to extend it to provide better support for techniques such as band-pass filters that require very little processing for their data-flow overheads. Investigation into an automatic mechanism for distributing the workload would also be an interesting and likely rewarding endeavour.

6. REFERENCES

[1] G. Tzanetakis and P. Cook. Audio information retrieval (air) tools. In *Proc. ISMIR*, 2000.

[2] G. Tzanetakis and P. Cook. Marsyas: A framework for audio analysis. *Organised Sound*, 4:30–??, 2000.

[3] J. Stephen Downie. The music information retrieval (mir) and music digital library (mdl) evaluation project, http://music-ir.org/evaluation/.

[4] Simple Utility Classes. sucs.sf.net/, 2004.

[5] J. Foote, M. Cooper, and U. Nam. Audio retrieval by rhythmic similarity. In *In Proc. ISMIR*, 2002.

[6] G. Wood and S. E. O'Keefe. Quantitative comparisons into content-based music recognition with the self-organising map. In *In Proc. ISMIR*, 2003.

[7] G. Tzanetakis, G. Essl, and P. Cook. Automatic musical genre classification of audio signals. In *In Proc. ISMIR*, 2001.

[8] Andreas Rauber and Markus Frühwirth. Automatically analyzing and organizing music archives. *Lecture Notes in Computer Science*, 2163:402–??, 2001.

[9] ed. P. R. Cook. *Music, Cognition And Computerized Sound: An Introduction To Psychoacoustics*. London, etc.,MIT, 1999.