FUNCTIONAL PEARL

Functional Programming as a Natural Choice for Spectral Centroid Analysis in Digital Signal Processing (DSP)

João Cândido Dovicchi
Center for Advanced Sonic Computing and Multimedia
Universidade Federal de Uberlândia - Uberlândia - MG - Brazil

João Bosco da Mota Alves
RExLab - Remote Experimentation Laboratory
Universidade Federal de Santa Catarina - Florianópolis - SC - Brazil

Abstract

This paper shows how implementing Functional Language Programming can be used to solve crucial problems with simplified methods. Working with large amount of data from digital signal, functions can be easily created to deal with statistical analysis of signal centroid. Importance of lazy evaluation and list comprehension in Functional Programming when dealing with lists and list of lists of experimental data that need to be automated is also shown. Results yielded from Functional Language were also compared to results yielded from imperative language (C routines).

1 Introduction

Many methods have been described to compress and encrypt sound signals (Quackenbush and Lindsay, 2001) and, in the last four years, the interest on IP telephony, videoconference over IP and other multimedia distribution as entertainment audio and video documents with Quality of Service (QoS) have increased the demand for research on those subjects (Hersent, Guide and Petit, 2000).

Efficient multimedia distribution over network is associated with reduced bandwidth consumption (Salami *et al.*, 1998). On the other hand, digital signal processing is related to appropriate low-pass filters that yield less quality lost during high frequencies cutoff to allow packets based multimedia comunication (Dovicchi and Barbar, 2001). The choice of filters that avoid such quality looseness and guarantee bandwidth economy can be related with analysis of centroid displacement and its stability on processed signals. As we have shown, this can be done through the average computation of spectral centroid on each defined frame of these signals (Dovicchi *et al.*, 1999).

It is already known that some analysis in sonic and other harmonic dependent signals like seismology (Dziewonski, Chou and Woodhouse, 1981; Dziewonski and Woodhouse, 1983), astronomy (Du *et al.*, 1998) and satellite image processing (Martin, 1989), require spectral centroid determination for an accurate comparison between data of different events of the same kind which determine suitable signal processing that yields good QoS.

Spectral centroids can provide a lot of information not only about the signal itself but also about the quality result of its processed data before it is distributed over the network.

Average centroid of a signal (Beauchamp, 1982) is the average frequency, weighted (multiplied) by amplitude of a frequency-domain signal. Its calculation is done using spectral centroids of selected frames of a time-domain signal transformed by methods by Fourier, Constant-Q, Wavelets and the like. We can notate it as:

$$C = \frac{1}{p} \sum_{p=1}^{j} C_p \tag{1}$$

Where, C_p is the spectral centroid of the p-th frame. There are two basic ways to calculate spectral centroids. One is defined in absolute values (Kendall, 1986) and the other one in Hertz (Sandell, 1991). The first kind of centroid is defined as:

$$C_p = \frac{\sum_{n=1}^{i} F_n \cdot I_n}{F_1 \cdot \sum_{n=1}^{i} I_n} \tag{2}$$

and the second kind is defined as:

$$C_p = \frac{\sum_{n=1}^{i} F_n \cdot I_n}{\sum_{n=1}^{i} I_n} \tag{3}$$

where C_p is the spectral centroid, $[F_1 \cdots F_n]$ are harmonic frequencies and $[I_1 \cdots I_n]$ are their respective intensities.

Albeit centroid calculation is an average of intensity weighted frequencies (product of each frequency by its absolute intensity) of signal partials (harmonic components), studies of related signals can be achieved through statistical analysis of large amount of indexed data. Actually, since spectral centroid describes a frequency (or a fundamental frequency factor) for a given frame, its average can represent the consistency of digital data. These data can derrive from different frames of the same signal or the same part of different filtered signals (*e.g.* steady-state region).

In both cases, data can be organized in list of tuples of (frequency,intensity) and centroid calculation can be performed on these data collection. Besides that, the algorithm has to be as general as possible and independent on the size of indexed data.

This fact turns the choice for a Functional Language programming an obvious one due to its lazy evaluation characteristics and because imperative programming requires explicit memory allocation for the algorithm design.

2 Functional Language and the Problem

The algorithm of centroid calculation is a very simple one. Eventhough there are no complexities at all in its design, one can notice that it depends heavily on the amount of data to be computed. So, the problem is to know how much memory allocation is necessary and, also, how to distribute computation if parallelism is available.

Implementing Functional Programming algorithm can be justified by the way Functional Language deals with lists. The use of imperative language to solve this problem depends on arrays of data to construct those lists and has to have necessary amount of memory allocated for that purpose. Lazy evaluation in Functional Language as Haskell, Clean and other Functional Language allows lists construction on demmand.

In our labs, as we work with Wavelet Transform of sound signals, we need to stablish the best number of iterations that still maintains the spectral centroid in a level of cerca 1% of its energy variation. Analysis are performed in fixed windowed frames of each sample and, spectral centroid of each sample is calculated. After, we determine the average of centroids data. Another average is then performed for each set of frames. Since frames are taken from the same signal, harmonic intensities reffer to the same set of frequencies and we can use equation 4 to determine the average centroid, directly:

$$C = \frac{1}{p} \cdot \sum_{p=1}^{j} \left(\frac{\sum_{n=1}^{k} F_n I_n}{\sum_{n=1}^{k} I_n} \right)_p \tag{4}$$

and, once $F_{1,1} = F_{1,2} = \cdots = F_{1,p}$ for the same signal or, for different frames of the same signal we have the same set of spectral frequencies, $F_{n,p}$ do not depends on p and equation 4 can be written as:

$$C = \frac{1}{p} \cdot \frac{\sum_{p=1}^{j} \sum_{n=1}^{k} F_n I_{n,p}}{\sum_{p=1}^{j} \sum_{n=1}^{k} I_{n,p}} \tag{5}$$

It is clear that implementing this algorithm on an imperative language needs nested loops, while equation 5 shows that in a Functional Language it can be done using "list comprehension" to construct lists of tuples between the list of frequencies and each set of intensities (see Figure 2). Operations then can be carried on directly in these constructors (Hudak *et al.*, 1992).

For example, let x be a list of spectral frequencies:

$$x = [F_1, F_2, \cdots, F_n]$$

and $y_1, y_2, \cdots, y_n$ lists of their respective intensities for the frames $1, \cdots, n$:

$$\begin{aligned} y_1 &= [I_{1,1}, I_{1,2}, \cdots, I_{1,m}] \\ y_2 &= [I_{2,1}, I_{2,2}, \cdots, I_{2,m}] \end{aligned}$$

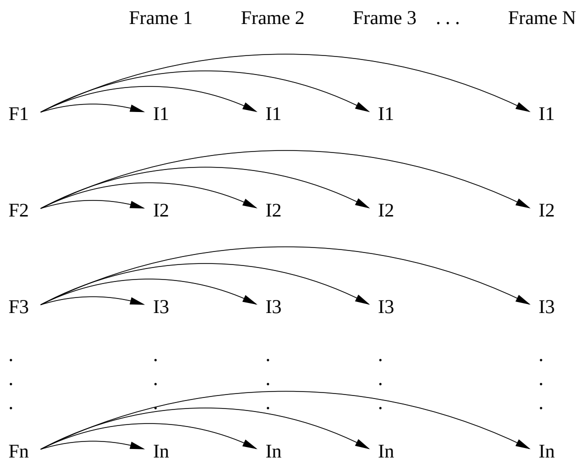

Fig. 1. Frequency and intensities tuples construction after each frame.

$$\begin{array}{rcl} \vdots & & \vdots \\ y_n & = & [I_{n,1}, I_{n,2}, \cdots, I_{n,m}] \end{array}$$

the list comprehension `[(f,c) | f <- Fn, c <- yn]` yields the list:

$$[(F_n, I_{n,1}), (F_n, I_{n,2}), ..., (F_n, I_{n,m})]$$

which can be easily contructed from experimental data for analitical purposes. In this case, working on a list of related tuples for each centroid of each sample is far easier than constructing all those loop iterations on an imperative programming language.

We can also justify the use of a Functional Language on the computationaly demanding tasks involved in such processing. It clearly exhibits an inherent parallelism. This parallelism may be exploited by an architecture providing a high degree of pipelining (Sérot, Quénot and Zavidovique, 1993).

3 Algorithm

Procedural algorithm for Kendall's centroid calculation is normally described as in figure 2.

Sandell's centroid definition is the same except that it returns only `swf / sa`. In this case centroid is expressed in Hz while Kendall's centroid returns an absolute value which is a factor of the fundamental frequency. It is easy to note that for great amount of data, there must be a great amount of memory allocation defined for the algorithm to compute data arrays.

The Functional Programming implementation proposed can be carried out in two ways. First, working with two distinct lists of frequencies and amplitudes (intensity) and, second using a list of tuples for $(freq, ampl)$. In both cases, parallel

```
let N = number_of_spectral_frequencies
for (i=0; i <= N; i++){
    swf += freq[i] * amp[i]
    sa  += amp[i]
}
return (swf/(freq[1] * sa))
```

Fig. 2. Algorithm for Kendalls spectral centroid

```
-- using two lists where x is a list of frequencies
-- and y is a list of its corresponding amplitudes.

module Centroid (centKendall, centSandell) where

centKendall :: (Fractional a) => [a] -> [a] -> a
centKendall x y = (swf x y) / (head x * sum y)

centSandell :: (Fractional a) => [a] -> [a] -> a
centSandell x y = (swf x y) / (sum y)

swf :: (Fractional a) => [a] -> [a] -> a
swf x y = sum (zipWith (*) x y)
```

Fig. 3. Module for functional implementation of spectral centroid algorithm, using two separate lists.

implementation can be easily deduced. In this paper, we call module **Centroid** the first type and module **Centroid_tp** (after "tuples") the second one. Both of them export two functions (centKendall and centSandell) for absolute and relative centroid calculations, respectively.

Figures 3 and 4 are listings of each module, where x is the list of frequencies, y is the list of absolute values of amplitude intensities and `swf` is the function that sums the weighted frequencies.

4 Test methods and results

Tests of these routines were performed on simple acoustic musical instruments signals which centroid data were already known. We begun with clarinet, horn and bassoon sounds. Centroid displacement where studied during Wavelet Transform using *Chroma4* algorithm (Dovicchi, 1999).

We have used the Glasgow Haskell Compiler (GHC), version 5.04, to implement those functions. When compared to the results obtained by C routines, functional modules yielded exactly the same results (see table 1).

FFT spectra results were grouped on a list of lists which have been transposed as

```
module Centroid_tp (centKendall, centSandell) where

centKendall :: (Fractional a) => ([a],[a]) -> a
centKendall (x,y) = (swf (x,y))/(head x * sum y)

centSandell :: (Fractional a) => ([a],[a]) -> a
centSandell (x,y) = (swf (x,y))/sum y

swf :: (Num a) => ([a],[a]) -> a
swf (x,y) = sum ( zipWith (*) x y)
```

Fig. 4. Module for functional implementation of spectral centroid algorithm, using a list of tuples.

	Recorded Sound		Chroma4 Transform (2 iterations)	
Samples	C-routine	Haskell	C-routine	Haskell
1	1.58518	1.58518	1.58204	1.58204
2	1.62538	1.62538	1.62848	1.62848
3	1.60371	1.60371	1.60990	1.60990
4	1.59442	1.59442	1.60061	1.60061
5	1.64705	1.64705	1.64086	1.64086
6	1.62643	1.62643	1.63010	1.63010

Table 1. *Bassoon steady-state centroid for original and transformed signal, using C routines compared to Functional routine (Haskell).*

described on figure 5[1]. As we always know the fundamental frequency, and number of frequencies on the obtained spectrum, we can construct a list of frequencies using `spectFreq` function. Each element of this list can be multiplied by its respective set of intensities and divided by sum of intensities values. Finally, the result is divided by the number of frames and returns the average centroid.

Listing in figure 6 is an example of how list of lists for weighted frequencies can be constructed. In this case, "listF" is the list of 4 harmonics of a 440 signal, "listI" is a list of lists of intensities (absolute amplitude) for respective frequencies and "listX" is the resulting list of lists of weighted frequencies. Actually, "listI" can be read from one or more files and we use other functions like `convData` and `argNum` listed on figure 5 to read intensities data from file and command line arguments. We

[1] Please, notice that the "transp" function is exactly the "transpose" function as described on Haskell's List Library. We declare it here for the sake of clarity.

```
module Cent_Tools (
   transp,
   convDat,
   spectFreq,
   argNum,
   mergeList,
   sliceList
   ) where

transp :: [[a]] -> [[a]]
transp []  = []
transp ([]:xss) = transp xss
transp ((x:xs):xss) = (x:[h | (h:t) <- xss]) : transp (xs:[t | (h:t) <- xss])

spectFreq :: (Enum a, Num a) => a -> a -> [a];
spectFreq n x = [ x | x <- map (x*) [1..n]]

mulFreq :: (Num a) => a -> [a] -> [a]
mulFreq x y = map (x*) y

mergeList :: (Fractional a) => [a] -> [[a]] -> [[a]]
mergeList [] _ = []
mergeList _ [] = []
mergeList x (y:ys) = zipWith (*) x y : mergeList x ys

convDat :: (RealFrac a) => String -> a;
convDat x = head (fst (unzip (readFloat x)))

argNum :: (Num a) => String -> a
argNum x = head (fst (unzip (readDec x)))

sliceList :: Int -> [a] -> [[a]]
sliceList _ [] = []
sliceList n x = fst (splitAt n x) : sliceList n (snd (splitAt n x))
```

Fig. 5. Module for functional implementation of tools for spectral centroid algorithm.

also defined a function `sliceList` because somtimes we have to slice a n-samples intensities data grouped on a single file.

Albeit we had noticed a loss of 40-50% in performance when functional implementation was compared to C, routines in C had to be manually adapted since spectral data was not allways the same length. Should we take this fact into account and the functional program can be considered quicker. Besides, it is far easier to deal with, for functions are clearly stated.

```
module Main where
main = do
      let listF = spectFreq 4 440.0
      let listI = [[0.9,0.6,0.8,0.4],[0.95,0.7,0.8,0.39],[0.92,0.67,0.79,0.42]]
      let listX= transp (mergeList listF listI)
      print (listX)
```

Fig. 6. Example for construction of a list of weighted frequencies lists referent to 3 frames.

5 Conclusions and Applications

Use of functional language has solved one main problem of our research. Since we work with signals of different lenght in time-domain and, many of them are windowed in thousands of frames, it is seldom necessary to rebuild the procedural routines. As we were working with C routines we could easily get out of memory, even if we did not run out of memory for real. To solve this problem we have written macros to deal with *lint* in order to make it stop complaining about *malloc, calloc, realloc*, and *free* while at the same time, performing checks of their arguments (Dovicchi, 1999). All these operations have to be done by hand and are time consumming.

Although not exactly a "pearl" of functional programming, this paper shows in practice how important a functional solution can be. It shows, also, how easy it is to deal with statistical research on data results. Lazy evaluation and "list comprehension" concepts are crucial to solve problems of data input from different length sets with a more generalized routine (module).

We did not pay too much attention to benchmark performance because our main interest was centered on the process automation. On this subject we are implementing the same algorithm in Clean which is faster than Haskell, eventhough it is not an important spot. We should point also that we have not exausted the possibilities of optimization when using GHC during program compiling.

Functional Programming concepts are to be applied also to other fields of interest the we are working on. For example:

1. Vowell centroid displacement for speech recognition;
2. Voice signal real time processing for telephony over IP;
3. Multicast audioconference and videoconference over IP;
4. Centroid displacement as function of sampled interval;
5. Filtered signals quality control for compressing algorithms; and
6. Centroid analysis during wavelet lifting itereations for digital filters.

References

Beauchamp, J. W. (1982) Synthesis by Spectral Amplitude and 'Brightness' Matching Analyzed Musical Sounds, *Journal of Audio Engineering Society*, 30 (6), pp. 396-406.

Dovicchi, J. C. (1999) Novos Coeficientes Wavelets Baseados em Intervalos Musicais para Análise de Timbres de Instrumentos Acústicos, *Doctorate Thesis*, Universidade Federal de Uberlândia, Uberlândia, BR.

Dovicchi, J. C., Carrijo, G., Lima, L. (1999) Fourier Transforms and Wavelets: a Combined Technique for Musical Signal Analysis. *Proceedings of the Conference on Mathematical and Computational Methods in Music, Diderot'99*, University of Wien, Wien, pp. 13-26

Dovicchi, J. C. and Barbar, J. S. (2001) Saving Bandwidth and Supporting Cryptography in the Internet: An Application of Wavelet Filters. *IIIS/IEEE Proceedings on Communication Systems and Internet*, Vol. XII (2), pp. 411-415.

Du, Z., Zhang, Y., Weng, J., Song, L. (1998) Calculation of binary image centroid by using TMS320C25, *In*: Electronic Imaging and Multimedia Systems II, Liwei Zhou and Chung-Sheng Li (Eds.), *Proc. SPIE*, Vol. 3561, pp. 409-412.

Dziewonski, A. M., Chou, T.-A and Woodhouse, J. H. (1981) Determination of earthquake source parameters from waveform data for studies of global and regional seismicity, *J. Geophys. Res.*, 86, pp. 2825-2852.

Dziewonski, A. M. and Woodhouse, J. H. (1983) An experiment in the systematic study of global seismicity: centroid-moment tensor solutions for 201 moderate and large earthquakes of 1981, *J. Geophys. Res.*, 88 pp. 3247-3271.

Kendall, R. A. (1986) The Role of Acoustic Signal Partitions in Listener Categorization of Musical Phrase, *Music Perception*, 4 (2) pp. 185-214.

Hersent, O., Guide, D. and Petit J.-P. (2000) IP Telephony, (1st. Edition) Addison-Wesley, New York.

Hudak, P., Peterson, J. and Fasel, J. (1992) A Gentle Introduction to Haskell Version 98, *Technical Report*, Department of Computer Science, Yale University.

Martin, D., (1989) Mapping Population Data from Zone Centroid Locations, *Trans. Inst. Br. Geogr.*, 14, pp. 90-97.

Quackenbush, S., and Lindsay, A. (2001) Overview of MPEG-7 Audio, *IEEE Trans. on Circuits and Systems for Video Technology*, Vol. 11 (6), pp. 725-729.

Salami, R., Laflamme, C., Adoul, J. P., Kataoka, A., Hayashi, S., Moriya, T., Lamblin, C., Massaloux, D., Proust, S., Kroon, P. and Shoham, Y. (1998) Design and Description of CS-ACELP: a toll quality 8 kbit/s speech decoder, *IEEE Trans. on Speech and Audio Processing*, 6 (2), pp. 116-130.

Sandell, G. J. (1991) Concurrent Timbres in Orchestration: A Perceptual Study of Factors Determining "Blend", *Ph.D. Diss.*, Northwestern University.

Sérot, J., Quénot, G. and Zavidovique, A. (1993) Functional Data-Flow Architecture Dedicated to Real-Time Image Processing, *IFIP WG10.3 Working Conference on Architectures and Compilation techniques for fine and medium grain parallelism*, pp. 129-140, Orlando, FL, USA.