1. Distributed Detection of Selected Features in Data Streams Using Grid-class Systems

Mariusz Matuszek

Gdansk University of Technology,
Faculty of Electronics, Telecommunication and Informatics,
Computer System Architecture Department
e-mail: mmatusze@eti.pg.gda.pl

Abstract

This chapter describes basic methodology of distributed digital signal processing. A choice of distributed methods of detection of selected features in data streams using grid-class systems is discussed. Problems related to distribution of data for processing are addressed. A mitigating method for data distribution and result merging is described.

Keywords*: grid system, data streaming, DSP.*

Modern science and technology acquire and record enormous amounts of data. Some examples are recordings from radio telescopes registering spectral components of radio signals arriving from deep space, analyzed either with hope of detecting some irregularities in background noise leading to possible clues of presence of other intelligences, or trying to hear the echoes of events which took place in a distant past, when time was still young. Another, more down to earth example is analysis of data streams acquired and recorded by telecommunications equipment, where searches are made for specific phrases and keyed-in number sequences, or bulk automated transcripts from speech to text are being performed.

The main problem with analyzing all those recorded data streams is their enormous volume, far exceeding the processing capabilities of not only a single personal computer, but of most modern computational clusters. Therefore, attempts are being made to harness on a large scale the power of personal computers, using coordinating middleware frameworks like BOINC or COMCUTE, which create grid-like computational environments distributing data and coordinating its processing across millions of personal computers all over the world. Such environments offer cheap voluntary processing power, however, their architectures impose specific restrictions on types of distributed algorithms which can be run efficiently. Some of those issues are addressed in later parts of this chapter.

1.1. Basic DSP operations and algorithms

A recorded data stream (a signal) can be viewed as a function, which may contain information in some domain, where most frequently information is contained within time, frequency, space and wavelet transform domains. Because of prevalent digital acquisition and sampling of input such functions are discrete. The main goal of

processing of such functions is to extract information in one or more domains of interest.

Many digital signal processing (DSP) operations are based on a convolution operation. Convolution is an mathematical operation on two functions which, in effect, produces a third function, which is typically viewed as a modified version of one of the input functions. A generic convolution expression is specified as follows (1):

$$\int_0^t \varphi(s)\psi(t-s)ds, \qquad 0 \le t \le \infty \tag{1}$$

The convolution of functions *f* and *g* is written as *f* * *g* and is defined as an integral of the product of the two functions, where one of the functions is reversed and shifted (2):

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau = \int_{-\infty}^{\infty} g(\tau)f(t-\tau)d\tau \tag{2}$$

and the symbol *t* represents the domain of the function.

When functions *f* and *g* are complex-valued, discrete and are defined on the set **Z** of integers, the discrete convolution of *f* and *g* is defined as (3):

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m] = \sum_{m=-\infty}^{\infty} f[n-m]g[m] \tag{3}$$

When function g_N is periodic with period *N*, then for functions *f* for which $f*g_N$ exists the convolution is also periodic and is equivalent to (4):

$$(f * g_N)[n] \equiv \sum_{m=0}^{N-1} \left(\sum_{k=-\infty}^{\infty} f[m+kN] \right) g_N[n-m] \tag{4}$$

where the summation on *k* is called a periodic summation of function *f*.

In cases where g_N is also a periodic summation of another function *g* the convolution $f*g_N$ is known as a circular convolution of *f* and *g*.

When durations of *f* and *g* are non-zero and are limited to the interval [0, *N*-1], convolution $f*g_N$ reduces to (5):

$$
\begin{aligned}
(f * g_N)[n] &= \sum_{m=0}^{N-1} f[m] g_N[n-m] \\
&= \sum_{m=0}^{n} f[m] g[n-m] + \sum_{m=n+1}^{N-1} f[m] g[N+n-m] \\
&= \sum_{m=0}^{N-1} f[m] g[(n-m)_{mod N}] = (f *_N g)[n]
\end{aligned}
\tag{5}
$$

Convolution operation plays an important role in digital signal processing. Some examples of the areas of use are: edge detection in image objects, various image transforms (for example Gaussian blurring), enhancement of digital audio signals (for example addition of artificial reverberation, adding or cancelling of echoes, etc.), calculating of weighted moving average of signal, calculating an output signal of linear electronic circuits where impulse response of a circuit is given as a second function, etc.

1.1.1. Spectrum analysis

A very typical and usually performed DSP operation is a transformation from a time domain into a frequency domain by application of a Fourier Transform [2] on the processed signal. This transformation provides information about amplitude and phase of frequencies composing the signal spectrum. In order to transform the digital signal from a time domain to a frequency domain a Discrete Fourier Transform (DFT) is applied. In practice, a Fast Fourier Transform (FFT) algorithm is commonly used for this task. The DFT formula is given as (6):

$$
X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i\frac{2\pi}{N}kn} \tag{6}
$$

which transforms a sequence of N complex numbers x_0 to x_{N-1} into a sequence of other complex numbers X, where X_k contains amplitude and phase components of k-th frequency bin in signal's spectrum band.

Implementation of the DFT algorithm directly leads to a computational complexity of $O(N^2)$ while optimized FFT algorithms achieve computational complexity of $O(N \log N)$.

1.1.2. Frequency detection – a Groetzel algorithm

While the Fast Fourier Transform algorithm computes evenly across the bandwidth of the analysed signal, often this is not necessary. In cases, where signal analysis is only concerned with detection of a few specific, known frequencies, the Groetzel algorithm (published by Dr. Gerald Groetzel in 1958) [3] is of immense use due to its low computational complexity.

Given an input sequence *x(n)* the Groetzel algorithm computes a sequence *s(n)*(7):

$$s(n) = x(n) + 2\cos(2\pi\omega)s(n-1) - s(n-2) \tag{7}$$

where $s(-2)=s(-1)=0$ and ω is some frequency of interest. For a fixed frequency, the part $2\cos(2\pi\omega)$ can be computed in advance, leaving only one multiplication operation, one subtraction and one addition for each input sample. Applying a Z-transform gives (8):

$$\frac{S(z)}{X(z)} = \frac{1}{1 - 2\cos(2\pi\omega)\,z^{-1} + z^{-2}} = \frac{1}{(1 - e^{+2\pi i\omega}z^{-1})(1 - e^{-2\pi i\omega}z^{-1})} \tag{8}$$

The time-domain equivalent becomes then (9):

$$\begin{aligned} y(n) &= x(n) + e^{2\pi i\omega}y(n-1) \\ &= \sum_{k=-\infty}^{n} x(k)e^{+2\pi i\omega(n-k)} \\ &= e^{+2\pi i\omega n}\sum_{k=-\infty}^{n} x(k)e^{-2\pi i\omega k} \end{aligned} \tag{9}$$

and finally, assuming $x(k)=0$ for all $k<0$ it becomes (10):

$$y(n) = e^{+2\pi i\omega n}\sum_{k=0}^{n} x(k)e^{-2\pi i\omega k} \tag{10}$$

This leads to a trivial implementation for a general purpose processors, because only values of *s(n-1)* and *s(n-2)* need to be retained in variables. An example pseudo code may look as follows in Lst. 1.1.

Lst. 1.1. Groetzel algorithm implementation example

```
s1 = 0
s2 = 0
normalized_f = target_f / sample_rate
coefficient  = 2 * cos(2 * PI * normalized_f)
foreach sample, x[n]
    s  = x[n] + coefficient * s1 - s2
    s2 = s1, s1 = s
out = s2 * s2 + s1 * s1 - (coefficient * s2 * s1)
```

Groetzel algorithm is very useful when searching recorded transmissions for dialed-in numbers, because decoding of DTMF[4] (touch tone) signaling used in telecommunications on consumer equipment requires detection of only eight frequencies. Another application where this algorithm excels is in software decoding of FSK (Frequency Shift Keying) data transmissions, where the number of frequencies is usually only two.

1.2. Distributed signal processing issues

One of fundamental limitations of grid-like distributed computing environments based on heterogeneous, geographically distributed voluntary computing power is inability to efficiently run distributed algorithms on problems, which require strong boundary coupling (data exchange) between neighbouring data sets. This limitation is twofold: first of all, such algorithms do not scale well because of differences in computing power of individual cooperating nodes as well as different network latencies between them. In practice the slowest CPUs and network links will tend to limit the performance of the whole set of cooperating nodes. In some cases the first obstacle may be mitigated by dynamically scaling the amount of data to be processed inversely proportional to the overall node performance. However, this is only possible for computations which allow variations in data size between different nodes. The second obstacle is even more prominent and is a result of an architecture of today's Internet, where individual nodes are frequently hidden behind layers of firewalls and network address translations. This makes bi-directional communication between individual nodes practically impossible. There are firewall piercing techniques, which in some cases make such communication possible, however their discussion is beyond the scope of this chapter. Besides, they always come at a cost, be it the bandwidth or the necessity to use third parties – which again does not scale well.

Fortunately, most DSP algorithms operating on one-dimensional signals have no or very little data exchange at dataset boundaries and scale very well in distributed environments. However, there is still a boundary issue to address. If we examine carefully typical DSP algorithms described earlier there is a class of uses which we can generally describe as a form of pattern search. Whether we are trying to detect a presence of a certain frequency or a spectral anomaly from a known signal background, a presence of a specific spectral signature (like a sound of explosion) or even decode DTMF number sequences, all these tasks can be viewed as either a form of a pattern search or as applying a predefined pattern (a function) to a signal. Since patterns have non-zero length, some additional precautions in data and result handling must be taken. They are described in the following paragraphs.

1.2.1. Data partitioning

A typical scenario of a data stream containing several events of interest is illustrated in Fig. 1.1. The task of processing this stream is to detect and qualify the events appropriately. For example, if the events were DTMF tones, the task would be to detect them and assign to one of 16 possible DTMF digits.

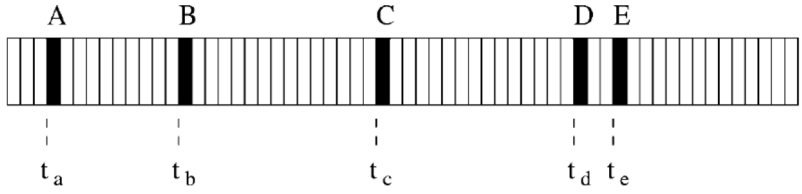

Fig. 1.1. An example stream of events A..E beginning at times t_a..t_e respectively

Fig. 1.1 shows five events, A-E, located on a time axis and beginning at times t_a to t_e respectively. Each of the events shown has a duration expanding over multiple samples (*n* samples), as given in Fig. 1.2.

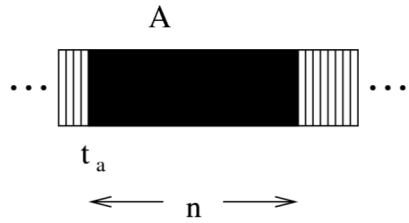

Fig. 1.2. A single event has a duration lasting multiple samples (n)

Processing the sample stream from a Fig. 1.1 on a uniprocessor machine with a suitable detection algorithm should yield a perfect detection of all five events, as illustrated in Fig. 1.3.

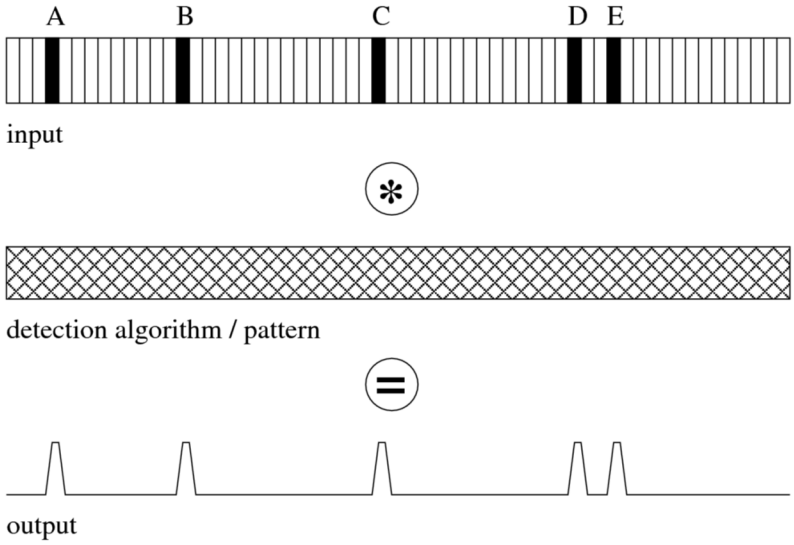

Fig. 1.3. A result of a perfect detection

It should be noted, that for most non trivial applications detection process requires more than a single sample to succeed. Therefore the output of a detection algorithm resembles more a trapezoid wave than a square wave. The leading and trailing edges of each trapeze represent areas of detection uncertainty.

Things become more interesting, when the same detection is attempted on multiple processing nodes, where parts of input data need to be distributed among all participants. A divide and conquer approach, which consists of dividing input into smaller chunks and distributing them among participating nodes is often used in such cases. However, a trivial application of a divide and conquer approach to data partitioning is almost guaranteed to produce detection artifacts, because in general it is impossible to know in advance whether a data split will occur inside or outside the set of samples constituting the event to be detected. The process, together with possible detection outcomes is shown in Fig. 1.4.

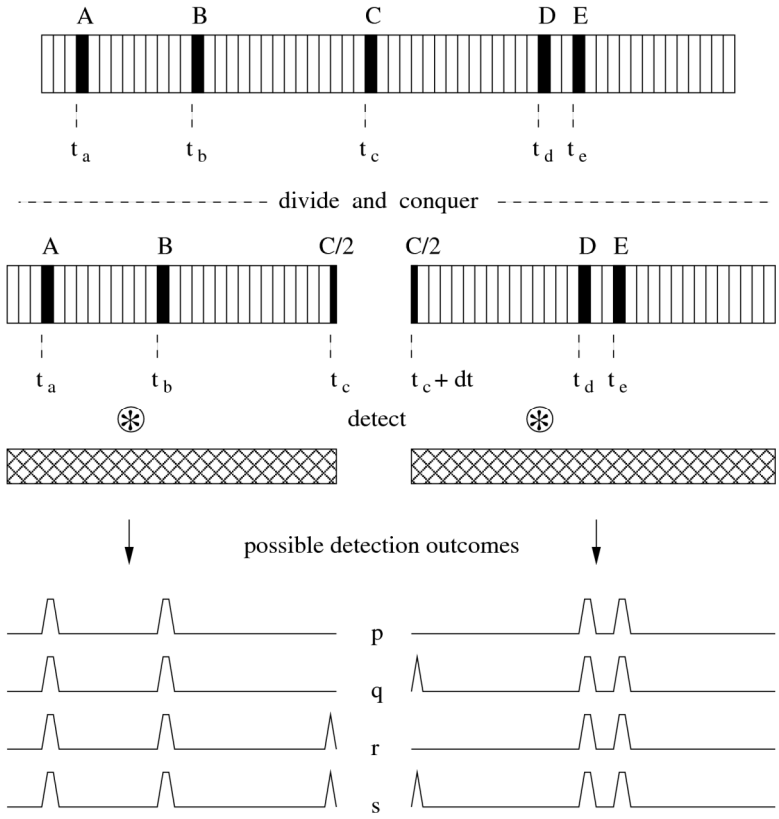

Fig. 1.4. Possible detection outcomes (p)..(s) with naive divide and conquer use

Fig. 1.4 shows a simple case of dividing data into two parts for distribution to two computing nodes. It can be observed, that the data boundary happened to be somewhere in the middle of samples constituting event C. Depending on properties of the detection algorithm used, four detection outcomes p, q, r, s are possible.

In outcome (p) neither of the computing nodes detected the C event.

In outcomes (q) and (r) one of the nodes happened to notice the C event, however with a marginal performance. Depending on how big the partial detection response is, the event may be accepted or rejected.

Finally, in outcome (s) both nodes marginally detect the C event.

It is easy to observe, that cases (q, r) will produce the correct detection result, albeit by chance and not by design. In cases (p) and (s) the result produced will either completely skip the C event or will have it duplicated. Neither of the two is correct of course.

To rectify this situation two actions are necessary. First, a small modification of the data dividing algorithm is required. By introducing some overlap between the data sets being distributed, the problem of false positives and false negatives can be addressed. This approach is presented in Fig. 1.5.

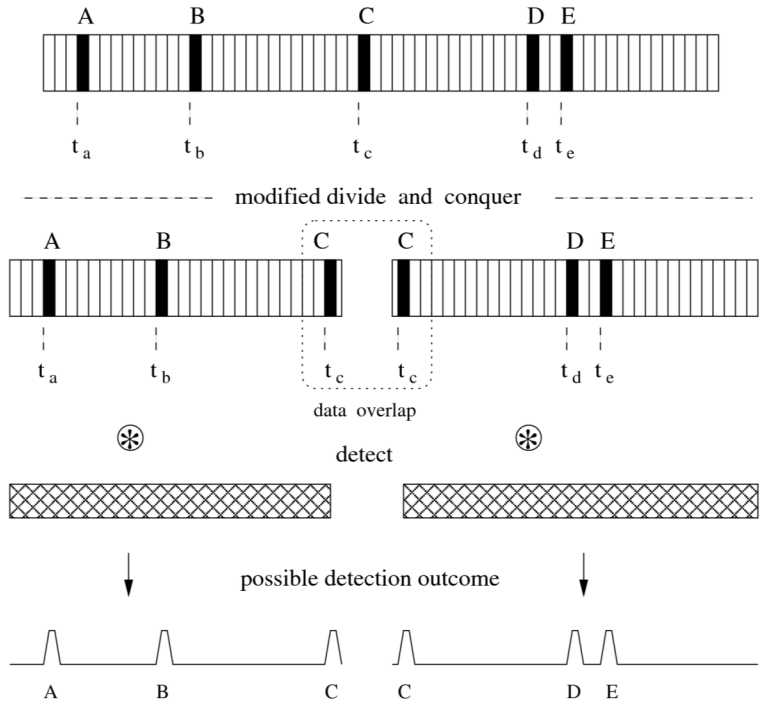

Fig. 1.5. Detection result when data overlap is applied

The size of the data overlap between data sets needs some attention and tuning because it depends largely on the nature of events being detected and on the cycle length of convolution operation (if convolution method is applied) or other pattern being used. With events of a uniform size the overlap may be minimal and equals the expected event duration (for ideal detection algorithms, in practice some small additional overlap will need to be applied). With events of variable length but with known size boundaries the safest overlap will be equal to the size of the longest event expected plus a possible small margin. Finally, with events of random length, the method described will only mitigate the possibility of false negatives and positives and the solution nature turns from deterministic to stochastic. Because overlapped data is in effect processed twice, the size of an overlap will have impact on the total performance.

It should be noted, that while overlapping the data solves or mitigates the discussed problem, it now creates another issue of its own. In Fig. 1.5 it can be observed, that all events are correctly detected, but the C event is reported twice. This is an expected side effect and must be addressed. This is best done in the result merging phase, which is described next.

1.2.2. Result merging

In a result merging phase it is necessary to discover duplicate detections of a single event and merge them back into a single result. This is trivial to achieve provided a single necessary condition is met. The condition is for all the processing nodes to utilize a common time base for the time axis of the input data. When this condition is met, each detected event can be time stamped by the node on which it was detected, which makes the duplicate merging problem easy.

It is well known in the field of distributed systems, that trying to depend on a common time base between distributed nodes is asking for races and troubles. Fortunately, in this particular case a vector time constructed from a sample number in an input sample sequence can be used, which makes the whole solution self-clocking and feasible. A possible approach to calculate time stamps for detected events is presented in Fig. 1.6.

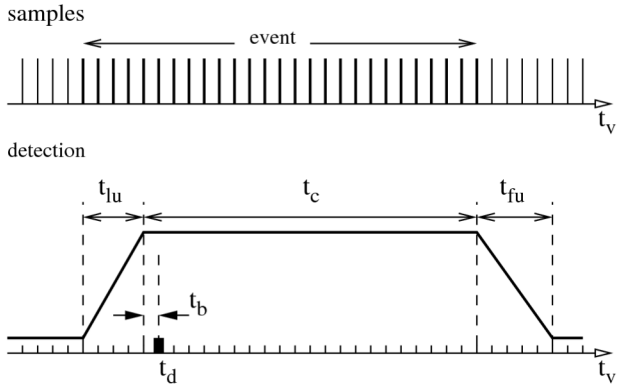

Fig. 1.6. Calculating the time stamps for detected events

Samples are distributed uniformly on t_v axis (vector time derived from a sample sequence number). Once the leading uncertainty time t_{lu} passes the detection algorithm is free to pick any point from the t_c (detection certainty) time range as a final time stamp t_d for the detected event. It is suggested that a small buffer delay t_b is introduced in order to avoid false positive detections. Once t_c passes the detection phase enters the t_{fu} region, which again is not a suitable candidate for placing of a detection time stamp.

As it was shown, with small modifications to data partitioning and to result merging schemes it was possible to rectify or at least mitigate the problem of false detections in the discussed example of distributed digital signal processing.

It should be noted that Comcute platform [6] provides users with the possibility to insert their own data partitioning and result merging algorithms into the system and call upon them in their computational tasks, which offers great platform flexibility enabling its adaptation to a wide range of use cases.

References

1. Sobolev, V.I.:, *Convolution of Functions*, in Hazewinkel and Michiel: Encyclopedia of Mathematics, Springer, 2001.
2. Bracewell, R.: *The Fourier Transform and Its Applications* (2nd ed.). McGraw–Hill, 1986.
3. Goertzel, G.: *An Algorithm for the Evaluation of Finite Trigonometric Series.* American Mathematical Monthly 65 (1): 34–35, 1958.
4. CCITT Vol. VI, *Recommendation Q.23*.
5. International Telecommunication Union: *ITU-T Recommendation F.902*, 1995.
6. Matuszek M.: *The Comcute system architecture,* in *Distributed computations in grid architecture systems*, Balicki J., Kuchta J. eds., pp. 83-92 , 2012, in Polish.