A FAST PROBABILISTIC MEDIAN ALGORITHM FOR INTEGER ARITHMETICS

Kari-Pekka Estola and Risto Suoranta

Machine Automation Laboratory, Technical Research Centre of Finland
P.O. Box 192, SF-33101 Tampere, Finland, Tel +358-31-163602, Fax +358-31-163494

Ce papier introduit une nouvelle méthode pour calculer la médiane mouvante d'une façon efficace. Le cas le plus mauvais de complexité du filtre proposé pour la médiane mouvante est de l'ordre d' $O(L)$, c.à.d linéairement proportionnel à la longueur de mot L. La complexité de la médiane mouvante ne dépend pas du nombre d'exemplaires inclus dans le calcul de médiane. Aussi, le nombre des placements de mémoire requis pour calculer la médiane est bas, étant de l'ordre de 2^{L+1} au pis. De plus, la structure de de contrôle du nouveau filtre de médiane est simple et facile à mettre en application dans la VLSI. L'algorithme se base sur l'estimation multiresolutionnelle de l'histogramme mouvant. Ainsi l'algorithme proposé peut être utilisé pour trourer plusieurs pourcentiles et pour estimes des modes et d'autres paramétres de données. Parce que la fenêtre mouvante peut être arbitrairement longue, la variation d'estimations pour les pourcentiles peut être basse. L'algorithme élémentaire donné dans ce papier fait les calculs pour la médiane et autres pourcentiles exactement, mais l'algorithme peut aussi être utilisé à calculer les approximations des vraies valeurs.

In this paper, we introduce a new method for computing the moving median efficiently. The worst case complexity of the proposed moving median filter is of order $O(L)$, i.e. linearly proportional to the wordlength L. The complexity of the moving median does not depend on the number of samples included in the median computation. Also, the number of memory locations required for computing the median is low being at worst of order 2^{L+1}. Furthermore, the control structure of the new median filter is very simple and it is easily implementable in VLSI. The algorithm is based on the multiresolution estimation of the moving histogram. Thus, the proposed algorithm can be used to find various percentiles and to estimate modes and other parameters of the data. Because the size of the moving window can be arbitrary long, the variance of the estimates to the percentiles can be made low. The basic algorithm given in the paper computes the median and other percentiles exactly but the algorithm can be also used to compute approximations to the true values.

1 Introduction

Median filtering has been widely used in digital image and signal processing for data smoothing and noise rejection. Although the median operation is nonlinear, theoretical studies and experiments have revealed some of its fundamental properties which are valuable in data processing. Median operation is attractive in applications where we want to filter noise from signals and sharp discontinuites have to be preserved.

The computational complexity of median filters, however, increases rapidly along the number of data samples included in the median operation. Typically the computational complexity of sorting N samples is $O(N \log N)$. In image and signal processing, the computational complexity can be reduced using moving median filters which compute the median within a window of N samples which is sliding over the data points. The reduction in the required number of arithmetic operations is due to the fact that the windows are overlapping and thus the data within the next window position is already partially sorted in the previous median operations. In the past years, several authors have developed fast methods for computing the median in real-time image and signal processing applications (see e.g. [1,2]).

In this paper, we introduce a new method for computing the moving median efficiently. The proposed method takes advantage of the finite wordlength representation of the digitalized stream of continuous data as the methods described in [1,2]. The complexity of the resulting median algorithm does not depend on the number of samples included in the median operation, but on the number of bits L used in representing the samples. Moreover, the worst case complexity of the proposed moving median filter is of order $O(L)$, i.e. linearly proportional to the wordlength L. Also, the number of memory locations required for computing the median is low being at worst of order 2^{L+1}. The proposed moving median filter can be viewed as a generalization of the reference methods (see [1,2]). The main difference between the reference methods and the proposed *multiresolution histogram method* is the use of the hierarchical variable depth tree structure in sorting the data. This approach results in an extremely fast median filter being faster than for example the radix method of Ataman *et al.*. Furthermore, the control structure of the new median filter is very simple and thus it is easily implementable in VLSI.

The basic algorithm given in the paper computes the median and other percentiles exactly but the algorithm can be also used to compute approximations to the true percentiles. This can be done simply by decreasing the number of bits in representing the data samples within the tree structrure. The computation of the approximate percentiles is faster than the exact percentiles because the effective depth of the

tree is reduced. It is shown in the paper that the number of required memory locations can be further reduced by weighting the depth of the tree according to the number of samples in the nodes. The weighting can based either on *a priori* knowledge or adaptive estimation of the data. In this case the accuracy of the obtained percentiles depends on the estimated probability density function. The estimation of the probability density function with a multiresolution histogram is built in the basic algorithm and thus it does not require any additional computation.

This paper discusses both the theoretical and practical aspects of the proposed multiresolution histogram method and performance comparisons are made with other existing algorithms.

2 Basic Algorithm

This section describes the new algorithm for computing various percentiles of integer data. The succeeding sections discuss the performance of the *multiresolution histogram method.*

In digital image and signal processing the data representing continuous time world is sampled using analog-to-digital converters. For most applications, it suffices to represent the discretized data with $L = 12$ or $L = 16$ bits. In many cases, the data processing hardware uses fixed-point arithmetics although floating-point processors are becoming popular. Hence the algorithms manipulating the integer data can be designed to exploit the finite wordlength representation. This approach is especially useful in designing fast algorithms for finding various percentiles of the data in a moving window.

The proposed algorithm for finding various percentiles is based on the multiresolution estimation of the histogram of the data in a specified window of the length M. The estimated multiresolution histogram forms an n-ary tree having L/d levels excluding the root. The relation between n and d is $d = \log_2 n$. Thus, the total number of memory elements R needed for the nodes excluding the root is

$$R = \sum_{k=0}^{L/d-1} 2^{L-kd}. \qquad (1)$$

The number of memory elements increases with decreasing d such that the maximum $R_{max} = 2^{L+1} - 2$ is obtained with the binary tree, i.e. $d = 1$. Figure 1 illustrates an n-ary tree with $n = 4$, $d = 2$ and $L = 8$.

Because the extent of the window limits the number of data samples to M for the estimation of the multiresolution histogram, the tree contains also exactly M data samples. The tree is implemented is such a way that each node in the tree contains the total number of the data samples in its descendants. Thus the nodes at a certain level comprise the histogram estimate. The histogram at that level is quantized with $ilev \cdot d$ bits, where $ilev$ is the specified level.

As the window slides over the data the contents of the nodes are updated. This is accomplished by incrementing the contents by one for the new sample and decrementing the contents by one for the outgoing sample. In order to achieve a fast realization, the proposed approach uses the values of the data samples as the addresses for the nodes to be updated. Thus the data flow is controlled by the data

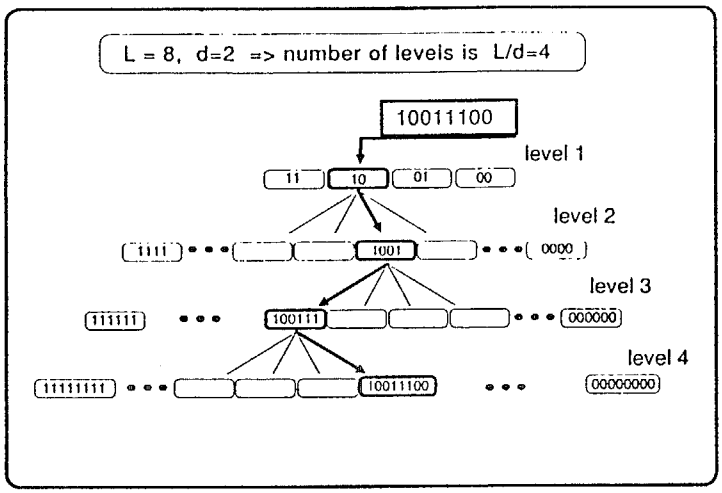

Figure 1: An example of the n-ary tree for the multiresolution histogram.

itself. The address for the node at the first level is the d most significant bits of the data and at the second level the $2d$ most significant bits and so on.

The various percentiles can be computed efficiently using the multiresolution histogram. The direction of the search for the percentile is from the root towards the leaves. At each level the search is performed from the lower addresses towards the higher addresses. The search is done by summing the contents of the nodes and comparing the sum to the value indicated by the specified percentile. When the value is exceeded the search continues on the next lower level. In the lowest level, the address of the node is the searched percentile.

An approximation to the percentile is obtained by stopping the search at an intermediate level $ilev$. The approximate percentile corresponds to the percentile of the data quantized to $ilev \cdot d$ bits. The computation of the approximate percentile is faster than for the exact percentile because the search path is shorter.

The estimated multiresolution histogram can be exploited in reducing the number of memory elements in the tree. In this approach, the resolution of the data within the tree is weighted according to the height of the histogram. This method assumes that the data occuring in the modes of the histogram should be preserved and the data falling inbetween the modes is less important. In this case, fewer nodes is used to store the data at the dips of the histogram. The amount of memory saved depends on the level where the tree is truncated. When the descendants of a node are removed at level $ilev$, the number of saved memory elements R_{save} is

$$R_{save} = \sum_{k=0}^{\widehat{L}/d-1} 2^{\widehat{L}-kd}, \qquad (2)$$

where $\widehat{L} = L - ilev \cdot d$. The adjustment of the depth of the tree can be done either based on the *a priori* knowledge of the probability density function of the data or adaptively.

3 Computing the histogram

In the case of sampled data the probability density function of the process from which the data is sampled is usually es-

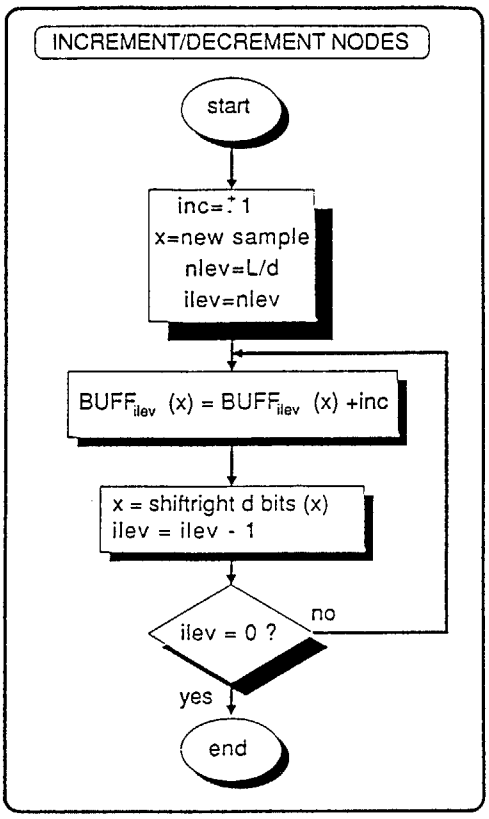

Figure 2: Flow control diagram for the tree maintenance procedure.

timated using *histogram* $h(n)$. In the *histogram*, the data is grouped into K class intervals of width W and $h(n)$ describes the frequency of samples falling in a specified interval n. The width W and the number of intervals K are chosen so that all possible samples belong to some interval in $h(n)$. In the case of fixed length integer data, the maximum resolution is achieved by choosing the width $W = 1$.

In the proposed algorithm for computing various percentiles, the first task is to construct the *multiresolution histogram*. In the *multiresolution histogram* several histograms are computed from the same set of data with different class interval widths W. In the bottom level the width $W_{nlev} = 1$ corresponds to the maximum resolution. In the next level $W_{nlev-1} = 2^d$ and, in general, W_{ilev} can be computed according to

$$W_{ilev} = 2^{(nlev-ilev)\cdot d}, \tag{3}$$

where *ilev* is the specified level of the histogram, *nlev* is the number of levels in the multiresolution histogram. The parameter d is determined by the ratio $L/nlev = d$, L being the number of bits used to represent the data.

The histogram is computed from the set of M data points, where the M is the length of the data window. When a new sample arrives, it is included to the data set and the oldest value is removed. A convenient way to handle the data set is to store it in a ring buffer. Besides the data set maintenance, the n-ary histogram tree structure must be updated for each new data sample.

The tree updating has two phases: in the first phase, the old value will be exported out of the tree and in the second phase, the new value will be imported into the tree. Removing a sample out of the tree means decrementing the nodes by one at each level of the tree and adding a new sample means incrementing the corresponding nodes by one.

Because of the simple structure of the tree, the index of the node to be updated can be extracted easily from the data itself. In the bottom level, the value of data sample is the index of the corresponding node. The index in the successive levels is obtained by shifting the previous level index d bits right and forgetting the outshifted bits. In the top level, the index is the d most significant bits of the processed data sample. Figure 2 presents the control flow diagram for the tree maintenance procedure.

Due to the low complexity operations and simple control structure it is easy to implement efficient realizations for the procedure.

4 Finding the percentiles

The percentile x_p for the probability distribution function $p(x)$ is defined as

$$\int_{-\infty}^{x_p} p(x)dx = percentile. \tag{4}$$

Various percentiles have specific names; x_{p50} is the median and x_{p25} is the lower quartile and so on. In the case of sampled data the histogram is used to aproximate the probability density function of the given set of data. Using histogram, the definition of x_p can be expressed as

$$\sum_{n=1}^{n_p} h(n) \geq percentile \cdot M, \tag{5}$$

where M is the number of samples and the index n_p is pointing to the interval enclosing the percentile x_p.

For the fast computation of percentiles it is useful to have only few intervals in the histogram, but for large W the histogram has a poor resolution. In the method proposed in this paper we combine the benefits of low and high resolution histograms to find percentiles efficiently. The percentile search is started from the top of the tree with the low resolution histogram; when the top level interval is found the search is continued in the next level. Depending on which level the search is stopped, different resolution estimates for percentiles are obtained.

Figure 3 depicts the flow control diagram for percentile search procedure. The length of the search path for some x_p depends besides on percentile itself but also on d chosen for the histogram tree. If $d = 1$, the n-ary tree becomes the binary tree. For the binary tree the search path is equal long for all x_p; in each level only one summing is needed to decide the right interval for the current percentile. When d is increased, fewer levels in the tree is needed but more summing has to be done before the right node is reached.

Figure 4 shows three curves describing the number of operations needed to update the histogram and to find a certain node on the bottom level. The worst case situation is in the end of the curves corresponding to the percentile lying in the last histogram interval. From the curves we can see that the most efficient percentile search is achieved by choosing $d = 2$. It is also seen that for $d = 1, 2$ the curves are quite flat, i.e. not much improvement is achieved by starting the search from a specific upper level node.

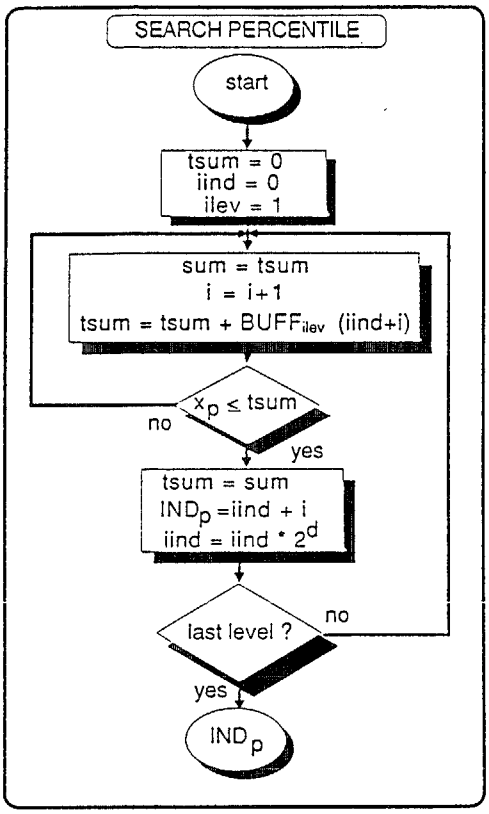

Figure 3: Flow control diagram for the percentile search.

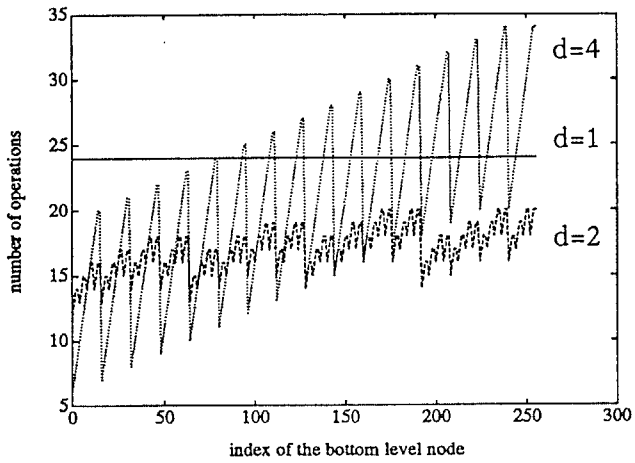

Figure 4: Number of operations for updating the histogram and finding a certain bottom level node; $d = 1, 2, 4$ and $L = 8$.

method	q=1	q=5	q=10	q=20	memory
Histogram	259	275	295	335	256
Radix	40	168	328	648	256
Proposed					
d=1	24	88	168	328	510
d=2	20	52	92	172	340
d=4	34	50	70	110	272
d=8	257	265	275	295	256

Table 1: Performance comparison ($L = 8$).

5 Comparisons

This section compares the performance of the proposed *multiresolution histogram method* with the histogram method of Huang *et al.* and the radix method of Ataman *et al.* in computing the moving median.

The main difference is that the proposed method has a variable depth tree structure whereas the reference methods use fixed data structures. Another notable difference is that the reference methods use more complex methods in computing or keeping track of the median. It also seems that the computation of more than one percentile degrades the performance of the reference methods more than that of the proposed method.

Table 1 depicts the number of operations involving memory increments and decrements, and comparisons. The equation giving the number of operations per computed median in the *worst case* is for the method of Huang *et al.*

$$O_H = 2^L + 4q - 1, \tag{6}$$

and for the method of Ataman *et al.*

$$O_A = L(4q + 1), \tag{7}$$

and for the proposed method

$$O_P = \frac{L}{d}(2^d + 2q - 1). \tag{8}$$

In the above equations q is the decimation ratio between the ingoing samples and the outcoming medians.

Table 1 shows that the proposed method is superior to the reference methods in terms of the computation complexity. The multiresolution histogram method is twice as good as the radix method for $q = 1$ and becomes even better as the decimation ratio q increases being approximately six times better for $q = 20$. The required number of memory elements is, however, somewhat higher for the proposed method at small values of d but at large values of d the difference is insignificant. Interestingly, the histogram method of Huang *et al.* is also better than the radix method for high values of q.

6 References

[1] T. S. Huang, G. T. Yang, and G. Y. Tang, "A Fast Two-Dimensional Median Filtering Algorithm," *IEEE Trans. Acoust., Speech, Signal Processing*, Vol. **ASSP-27**, pp. 13–18, Feb., 1979.

[2] E. Ataman, V. K. Aatre and K. M. Wong, "A Fast Method for Real-Time Median Filtering," *IEEE Trans. Acoust., Speech, Signal Processing*, Vol. **ASSP-28**, pp. 415–421, Aug., 1980.