Fast median calculation method

J. Cadenas, G. M. Megson, R. S. Sherratt and P. Huerta

The ever increasing demand for high image quality requires fast and efficient methods for noise reduction. The best known order-statistics filter is the median filter. A method is presented to calculate the median on a set of *N* *W*-bit integers in W/B time steps. Blocks containing *B*-bit slices are used to find *B*-bits of the median; using a novel quantum-like representation allowing the median to be computed in an accelerated manner compared to the best known method (*W* time steps). The general method allows a variety of designs to be synthesised systematically. A further novel architecture to calculate the median for a moving set of *N* integers is also discussed.

Introduction: The median filter is the most common non-linear spatial filter known for its excellent noise-reduction capability in smoothing of images [1]. It also finds applications broadly in digital signal processing analysis [2]. The median, *M*, on a set of integers is such that half the integers in the set are *less or equal* to *M*, and half are *greater or equal* to *M*. Without loss of generality, an odd number of integers in the set, $N = 2k + 1$, is most practical; this is maintained throughout this Letter. If the integers were sorted, the median is the integer in the middle location.

The median can be calculated by sequential sorting with a complexity of $O(N \log N)$ [3]. Non-sorting based methods, especially those designed for hardware architectures achieve the calculation in a number of steps related to the bit length of unsigned integers, *W*, rather than *N* [4]. The method of Prokin and Prokin [5] is better than previous sorting and non-sorting methods and uses *W* steps to find the median. This Letter presents a method to calculate the median on a set of *N* *W*-bit integers in

W/B time steps, where B is a parameter used to decide how to slice the integers in the data set into blocks.

The new method creates slices of B-bits, forming W/B processing blocks, each block calculating B-bits of the output median. Blocks of 2-bit or 3-bit are practical for hardware implementation. Working on more than 1-bit reveals hidden parallelism. However, no previous known method operates on more than 1-bit at a time, as presented here. The data representation of B-bit slices is changed on the fly allowing the method to explore ahead, further than one bit a time, using a Quantum Representation (QR) of bits, which retains the advantages of flexible data manipulation and data parallel properties. This Letter starts with an example on a small set of non-negative integers. The analysis produces an architecture for calculating the median on a sliding window of size N. The method is then extended to cover signed integers and to solve the more generic problem of a rank filter.

Quantum representation: Define a data bit as a *qubit* with the form $d = a_0Q_{[0]} + a_1Q_{[1]}$ where a_i are quantum amplitudes [6]. The qubit is a superposition of two states defined as $Q_{[0]} = [0\ 1]$ and $Q_{[1]} = [1\ 0]$. Hence, $d = a_0\ [0\ 1] + a_1\ [1\ 0] = [a_1\ a_0] = Q_{[d]}$; if $d = 0$, $Q_{[d]} = Q_{[0]}$, else $Q_{[d]} = Q_{[1]}$. The tensor operator onto two qubits is defined as: $Q_{[g]} \otimes Q_{[h]} = [g_1\ g_0] \otimes [h_1\ h_0] = [g_1h_1\ g_1h_0\ g_0h_1\ g_0h_0]$ which extends straightforwardly to higher dimensions. A p-bit binary string $X_j = x_{p-1}x_{p-2}\ ...\ x_0$ can be represented as $Q_{[x_j]} = Q_{[x_{p-1}]} \otimes Q_{[x_{p-2}]} \otimes \ldots \otimes Q_{[x_1]} \otimes Q_{[x_0]}$, which forms a 2^p-bit QR. For instance, when $W = 2$, $x_1x_0 = 00$, $Q_{[00]} = [0\ 1] \otimes [0\ 1] = [0\ 0\ 0\ 1]$. This format maps the input bits to a set of mutually orthogonal vectors which allows implicit parallel processing and provides a unique index within a 2^p elemental vector.

Small example: Consider a data set of $N = 5$ elements 4, 8, 2, 7, 14, each of $W = 4$ bits. $N = 2k+1$ implies $k = 2$ and indicates that, $P = k+1 = 3$ is the position of the median. Partitioning the input data with B = 2 bits forms two groups as in Table 1, creating vertical slices of two bits (far left in table). Group 1 is processed first. The QR is generated, for each 2-bit slice and labelled as q_i for i = 3, ..., 0. All q_i are added in parallel (vertically). These sums are then accumulated right to left into A. A parallel comparison is performed on each accumulation against P, searching for the first occurrence (right to left) when $A \geq P$; (column q_1 in the example). The two MSB of the median are then found as “01”, indicating that only the elements 4, and 7 are median candidates. Next, group 2 is processed. First, P is recalculated as P = 3 -1 = 2 (1 being the A value to the right under q_1 column) and then the slices for elements 8, 2 and 14 (gray in table) are nullified. Computing the QR, sum and accumulation for the second group proceeds, as before, on the remaining 2-bit slice. The condition $A \geq P$ is first satisfied for A under q_3. The remaining two bits for the median are “11”. Combining the results from group 1 and 2 give the median as $M = 0111 = 7_{10}$.

The QR representation produces a virtual parallel sorting mechanism (see blocking 4-bits at the time, bottom of Table 1). For 2-bit blocks, block 1 discards item elements 2, 8, and 14. The element 2 must be to the right of the median, so the median new position is P = 3 – 1 = 2, within the only remaining median candidates 4 and 7. The process generalises to large problem sizes and in principle after the accumulation and comparisons, the median can be obtained in a single step. The exponential nature of the tensor operator for generating QR becomes unwieldy for large W, but for small and reasonable cases (particularly those used in applications) can provide a fast architecture.

Sliding window median architecture: Figure 1 shows an inside view of block 1 to process a set of N elements of W bits each, by blocking B-bits at a time which produces $\lceil W/B \rceil$ processing blocks. Input elements are pipelined with N stages. Each stage holds B bits; this forms a sliding window with element slices $X^1_{W-1-(j-1)B}$, ...,

X^N_{W-jB}, for $j = 1, 2, ..., W/B$. The QR is generated for each slice producing 2^B bits with indices $i = 0, ..., 2^B-1$. The QR is easily implemented with B-to-2^B binary decoders. A parallel sum of N bits each is performed along index i and labelled S^1_i for block 1 with accumulation A^1_i. For the sum S, carry-save-adder (CSA) tree structures [7] produce a sum of $\log_2 N$ bits over which the accumulation is performed. For a typical $N = 9$ or 11, fast 4-bit adders will suffice. The expected position for the median is P^1 for the first block with a value $P^1 = k+1$. Internally to the logic, 2^B parallel *greater than or equal* comparisons are performed on the $\log_2 N$ bits of A^1 against P^1. A simple arrangement of XOR gates then extracts the first occurrence of the comparisons $A \geq P$. A single bit over an i bit-array, $i = 0, ..., 2^B-1$ will be set (for $B = 2$, this could be [0 0 1 0]) as the result of the comparison process. The set bit position bit-code corresponds to the B bits from the median that can be extracted in that block. The logic then calculates the new position P^2 to be used in the next block, calculated as $P^2 = P^1 - A^1_{i-1}$ for the i value where the comparison is true.

All previous results are pipelined vertically where the calculation continues to the next block. Elements that cannot be the median get nullified (using the enable bits shown in the figure as e^1_t, $t = 1, ..., N$ for the first block). A structure to generate these nullify enable signals for the next block is shown in Figure 2 (left) along with a suggested logic for each E_t block in the figure (for $B = 2$, right). For the first block these enable signals are conveniently wired to 1. Thus, after W/B processing blocks the median M is found at the bottom, with each block contributing B bits of M. The position of the median in the window is indicated by the enable set bits generated on the last block.

Timing analysis: Reducing the median computation by B steps would not represent an improvement if the time of each step, due to the complexity of the arrangement, is larger than before. Figure 3 shows that this does not happen for $N > 7$. The median computation time in [5] is $t_{[5]} = WT_{[5]}$ and, for figure 1, $t = (W/B)T$, thus the speedup factor of Figure 1 is $s = B(T_{[5]}/T)$ with $T_{[5]}$ and T representing the critical paths of [5] and Figure 1 respectively. At N = 15, directly from Figure 3, $T_{[5]}/T$ ~ 0.625 for B = 2, and consequently speedup s = $B(T_{[5]}/T)$ ~ 1.25 or 25% faster. Similarly, a 25% speedup is seen for B = 3. We corroborated this evidence for a circuit implemented in FPGA technology (*Virtex* 4 device) for the case N = 15 and B = 2. T is essentially due to the CSA tree followed by a chain of 2^B-1 accumulators of $\log_2 N$ width each, which make them suitable for implementation with fast full carry look-ahead adders [7]. $T_{[5]}$ is basically the delay cost of the CSA tree.

Signed integer extension: Let the number of positive integers be $C0$ and negative integers $C1$, then $N = C0 + C1$. If $C0 > C1$, set $P = k + 1 - C1$ initially, otherwise set $P = k + 1 - C0$; the method applies unmodified to the remaining W-1 bits.

A rank filter: An order R filter, for a set of N elements has R elements *less or equal* to the output and $N - R$ elements *greater or equal* to the output [5]. For $N = 2k + 1$, the median is a rank filter with $R = k$. The method in this Letter calculates the median by setting $P = R + 1$ initially. The accumulation in figure 1 can be performed left to right as shown, or right to left. Thus, in order for the method to perform as a rank filter, for left to right accumulation, requires setting $P = N - R$ initially. For right to left it requires setting $P = R + 1$.

Conclusion: This method reduces by B the number of steps to calculate the median on a set of N items compared to previously known methods. We found parameters B and N for which circuit implementations also make the method faster. This is ideal for real-time processing of images. The method applies to non-negative or signed integers and also easily extends to the case of rank filtering. Multiple architectures can be derived with different tradeoffs between area and time in fully pipelined structures.

References

1 GONZALEZ, R., and WOODS R.: ‘Digital image processing’ (Prentice Hall, 2002)

2 YOUNG, N., and EVANS A. N.: ‘Median centred difference gradient operator and its applications in watershed segmentation’, *Electron. Lett.*, 2011, 47, (3), pp. 178-180

3 CORMEN, T. H, LEISERSON, C. E., RIVEST R. L., and STEIN C.: ‘Introduction to Algorithms’ (The MIT Press, 2003)

4 CHANG, S., and CHIG, W.: ‘A parallel median filter with pipelined scheduling for real-time 1D and 2D signal processing’, *IEICE Trans. Fundamentals*, 2000, E83-A, (7), pp. 1396-1404

5 PROKIN, D., and PROKIN, M.: ‘Low hardware complexity pipelined rank filter’, *IEEE Trans. On Circ and Syst. II.*, 2010, 57, (6), pp. 446-450

6 NEILSEN, M. A., and CHUANG, I. L.: ‘Quantum computation and quantum Information’ (Cambridge, 2000)

7 PARHAMI, B.: ‘Computer arithmetic, algorithms and hardware designs’ (Oxford, 2000)

Authors’ affiliations:

J. Cadenas and R. S. Sherratt (School of Systems Engineering, University of Reading, Reading, RG6 6AY, United Kingdom)

G. M. Megson (School of Electronics and Computer Science, University of Westminster, London W1T 3UW, United Kingdom)

P. Huerta (Escuela Técnica Superior, Universidad Rey Juan Carlos, Madrid, Spain)

E-mail: o.cadenas@reading.ac.uk

Figure Captions

Fig. 1 A pipelined architecture of a block to calculate B-bits of the median by processing slices of B-bits from a set of $N = 2k + 1$ non-negative elements of W-bits.

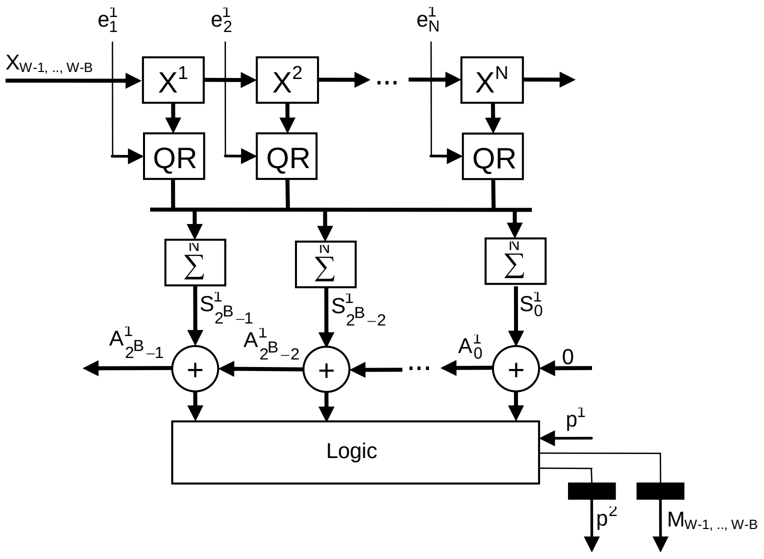
Fig. 2 Left: Nullify enable signal architecture and Right: A logic structure for each block E_t on the left for the case of $B = 2$ bits, $W = 4$ bits.

Fig. 3 Estimated ratio of $T_{[5]}/T$ for critical paths of the architecture in [5] ($T_{[5]}$) against the critical path of the architecture in Figure 1 (T) as a function of window size N.

Table Captions

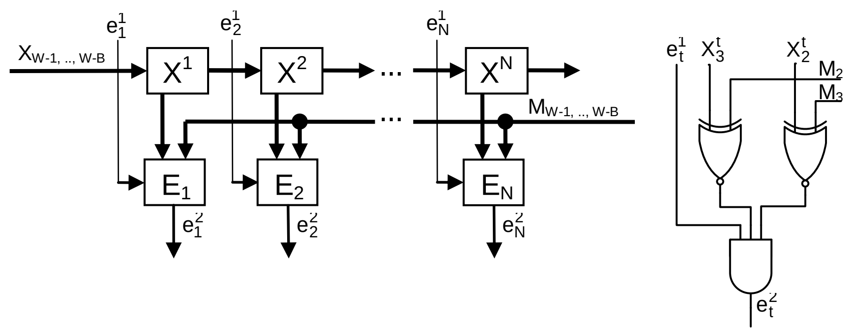
Table 1: (Top) Median calculation for input items 4, 6, 2, 7, 14 by blocking 2-bits at a time. (Bottom) Quantum representation of 4, 6, 2, 7, 14 by blocking 4-bits at a time.

Figure 1



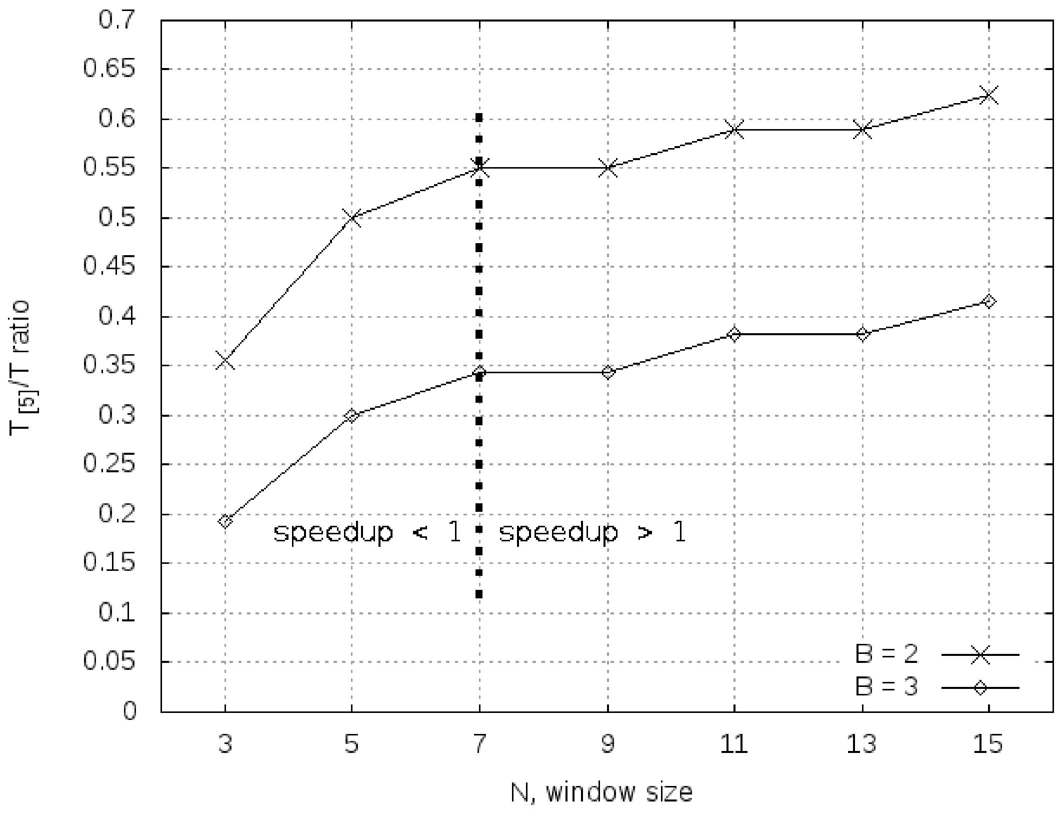
e_1^1
e_2^1
e_N^1
$X_{W-1,..,W-B}$
X^1
X^2
...
X^N
QR
QR
QR
$\sum^N$
$\sum^N$
$\sum^N$
$S^1_{2^B-1}$
$S^1_{2^B-2}$
S^1_0
$A^1_{2^B-1}$
$A^1_{2^B-2}$
A^1_0
0
+
+
+
...
Logic
p^1
p^2
$M_{W-1,..,W-B}$

Figure 2



e_1^1
e_2^1
e_N^1
$X_{W-1, .., W-B}$
X^1
X^2
X^N
...
$M_{W-1, .., W-B}$
E_1
E_2
E_N
e_1^2
e_2^2
e_N^2
e_t^1
X_3^t
X_2^t
M_2
M_3
e_t^2

Figure 3



0.7
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
$T_{[5]}/T$ ratio
3
5
7
9
11
13
15
N, window size
speedup < 1
speedup > 1
B = 2
B = 3

Table 1

$Item_{10}$	$Item_2$
4	0100
8	1000
2	0010
7	0111
14	1110

Group 1	q_3	q_2	q_1	q_0
01	0	0	1	0
10	0	1	0	0
00	0	0	0	1
01	0	0	1	0
11	1	0	0	0
$S = \Sigma q_i$	1	1	2	1
$A = \Sigma S$	5	4	3	1

Group 2	q_3	q_2	q_1	q_0
00	0	0	0	1
00	0	0	0	0
10	0	0	0	0
11	1	0	0	0
10	0	0	0	0
$S = \Sigma q_i$	1	0	0	1
$A = \Sigma S$	2	1	1	1

q_{15}	q_{14}	q_{13}	q_{12}	q_{11}	q_{10}	q_9	q_8	q_7	q_6	q_5	q_4	q_3	q_2	q_1	q_0
0	1	0	0	0	0	0	1	1	0	0	1	0	1	0	0