UNCLASSIFIED

AD-A277 060

AR-006-185

DSTO

Electronic Warfare Division

RESEARCH REPORT
ERL-0692-RR

PARALLEL PROCESSING FOR DIFFERENCING AND HISTOGRAMMING

by

Vaughan Clarkson and Pinaki S. Ra

94-08741

APPROVED FOR PUBLIC RELEASE

UNCLASSIFIED

ELECTRONICS RESEARCH LABORATORY

AR-008-338

DSTO AUSTRALIA

ELECTRONICS RESEARCH LABORATORY

Electronic Warfare Division

RESEARCH REPORT
ERL-0692-RR

PARALLEL PROCESSING FOR DIFFERENCING AND HISTOGRAMMING

by

Vaughan Clarkson and Pinaki S. Ray

SUMMARY

Algorithms which take advantage of massively parallel and vector architectures are formulated for the calculation of the differences of times-of-arrival derived from a sequence of time samples. The application which generated the work is the classification of pulse trains which are detected by an electronic support measures receiver.

Four algorithms are proposed, of which three have been implemented and tested on several machines. The results obtained through testing are compared with each other and with results for more conventional architectures. It is found that one of the machines, the massively parallel *MASPAR* MP-1 computer, is able to perform histogramming fastest through exploitation of its data-parallel architecture. Several suggestions are made which could significantly improve performance on all architectures.

OCT 93

APPROVED FOR PUBLIC RELEASE

POSTAL ADDRESS: Director, Electronics Research Laboratory, PO Box 1500, Salisbury, South Australia, 5108.

CONTENTS

TABLES

FIGURES

APPENDICES

THIS IS A BLANK PAGE

ABBREVIATIONS

ACU	Array Control Unit
ALU	Arithmetic Logic Unit
AOA	Angle of Arrival
ESM	Electronic Support Measures
MIMD	Multiple Instruction Multiple Data
PDW	Pulse Descriptor Word
PE	Processor Element
PRI	Pulse Repetition Interval
PRF	Pulse Repetition Frequency
RWR	Radar Warning Receiver
SIMD	Single Instruction Multiple Data
SISD	Single Instruction Single Data
TOA	Time of Arrival
TDOA	Time-Difference of Arrival

THIS IS A BLANK PAGE

1 INTRODUCTION

Passive detection and identification of radars in real time is a critical operational requirement for Electronics Support Measures (ESM) equipment such as radar warning receivers (RWR's). Typically, a receiver used in ESM intercepts signals from a number of radars and other emitters within its frequency band contemporaneously. In order to identify individual emitters from this composite signal, one radar parameter which is often derived is the pulse repetition interval (PRI) for each emitter, i.e. the interval between the times-of-arrival (TOA's) of consecutive pulses which it emits.

Other parameters which may be measured are the angle of arrival (AOA), frequency and amplitude to form a set of pulse descriptor words (PDW). Utilising the AOA data, the pulses could be presorted into clusters of adjacent angular sectors, each of which could be processed independently. However, the situation can arise in which it is impossible to distinguish emitters on these parameters alone because of inadequate resolution. It is then required to extract as much information as possible from the TOA data.

In a dense environment comprising many types of emitters, the processing of this information is critically time intensive, e.g. for real scenarios one could envisage a pulse data rate of 10 million pulses per second. Thus for real time operation, any improvement in the speed of processing is highly desirable.

Following this concept, the Information and Signal Processing Group of the Electronic Warfare Division has proposed the application of parallel processor technology for the study of deinterleaving of the pulse trains. The object of this paper is to outline the formulation of PRI processing, and in particular time-differencing and histogramming, in this context. The paper assumes that a buffer of TOA's is continuously filling with recorded pulses, and the buffer data is periodically processed (and the buffer emptied) by the processor(s).

In Section 2, time-difference-of-arrival (TDOA) histogramming is introduced. This is a common processing step in ESM equipment and it is useful because it highlights periodicities in the TOA data. Some of the properties of the TDOA histogram are discussed.

Several methods of performing differencing and histogramming on massively parallel architectures are proposed in Section 3. The method of implementation on a two-dimensional grid of processors is discussed, with particular reference to the *MASPAR* massively parallel computer. The requirements in time, memory and number of processors for each method is examined as a function of the number of TOA's in the buffer.

A method for calculating the TDOA histogram on a vector processor architecture is discussed in Section 4. Again, the requirements in time and memory are examined as a function of the number of TOA's.

In Section 5, the histogramming methods for massively parallel, vector and traditional processors are compared and timed. A *MASPAR* MP-1 is used to implement the massively parallel methods, whereas a Fujitsu VP-2200 is used to implement the vector processor methods. Sun and Hewlett-Packard workstations were used to implement histogramming on traditional processors. Timing comparisons are presented for various histogram bin widths and pulse densities and the results are reconciled with the theory.

Suggested improvements and future directions are outlined in Sections 6 and 7. In these sections, several suggestions are made that might improve the performance of the algorithms with respect to their speed of computation and areas of ongoing research are highlighted.

Finally, conclusions are drawn in Section 8 where it is found that the *MASPAR* MP-1 performs the fastest histogram of the architectures examined. However, no account has been made for the input/output time requirements of the machines examined, which in practice could significantly alter their performance.

2 TDOA HISTOGRAMMING

Histogramming is one of the most widely used processing operations for the detection of pulsed emitters [2,6–9]. Let the TOA data for N pulses from the PDW's be arranged in order of arrival as a vector t where

$$t = \begin{bmatrix} t_0 & t_1 & \cdots & t_{N_p-1} \end{bmatrix}^T \tag{1}$$

where N_p is the number of PDW's in the sample and $t_i < t_j$ for $i < j$.

The matrix of the differences in times of arrival, Δ, is then defined by its element Δ_{ij} where

$$\Delta_{ij} = t_j - t_i$$

where $0 \leqslant i, j < N_p$. Note that Δ is antisymmetric.

The TDOA histogram is the vector H of h_k (where the individual h_k are known as the "histogram bins") of dimension N_H such that

$$h_k = \#\{kt_b \leqslant \Delta_{ij} < (k+1)t_b, 0 \leqslant i, j < N_p\},$$

t_b is the histogram bin width, and $k = 0, 1, \ldots, N_H - 1$ and $\#\{\cdot\}$ represents the cardinality of the set.

The attractive property of the TDOA histogram is that for a pulse train of periodicity T, peaks will appear in histogram bins h_k where $k = \lfloor nT/t_b \rfloor$ for all integers $n > 0$ where $\lfloor \cdot \rfloor$ returns the largest integer smaller than its operand. Further, interference between pulse trains tends to produce a "flat" noise background. That is, bins which do not contain peaks caused by pulse trains or their harmonics have a relatively constant amplitude. It is therefore relatively easy to visually detect peaks in the histogram from the interference noise under most circumstances.

Automatic detection and estimation methods are currently topics of ongoing research, and fall outside the scope of this report [1]. This report concerns itself only with the calculation of the histogram.

Further, it is a non-trivial system design problem to decide on the appropriate buffer length for incoming pulses to histogram, the appropriate bin width for the histogram and how to then combine or interpret the results of consecutive histograms from the buffers. Again, this is considered to be outside the scope of this report.

3 DIFFERENCING AND HISTOGRAMMING USING THE SIMD ARCHITECTURE

The model considered in this section is the Single Instruction Multiple Data architecture (SIMD). In this computational model, a single instruction is issued, one at a time, to the complete set of processors. Each active processor performs this instruction on its own local data. This is in

contrast to the Single Instruction Single Data (SISD) model or von Neumann model (used in traditional computers such as personal computers and most workstations) and the Multiple Instruction Multiple Data (MIMD) model in which each processor can be executing separate instructions on separate data.

In particular, a SIMD model is examined in which the processors are arranged in a two-dimensional matrix, with each processor element able to communicate data with any other processor element, and each having both local (private) and globally shared memory. This is the architecture of the *MASPAR* computer, on which numerical simulations were performed as described in Section 5.

Each element of the matrix is an individual processor, denoted μ_{ij}, $i = 0, \ldots, N_y - 1$; $j = 0, \ldots, N_x - 1$, where N_x is the number of columns of the matrix and N_y is the number of rows. The total number of processors is therefore $N_\mu = N_x N_y$. It is also possible to "unwrap" the matrix of processors and consider the processors as a linear array. In this case the processors can be denoted μ_i, $i = 0, \ldots, N_\mu - 1$.

Three methods are now proposed to perform differencing of TOA's on this SIMD architecture. The first method is designed to calculate the matrix of time differences, whereas the latter two are designed for histogramming.

3.1 Method 1

This method aims to produce the TOA difference matrix, Δ. It does not calculate the TDOA histogram, but it is a straightforward operation to collate the matrix into the histogram. It is a nearest neighbour algorithm, which gives a speed advantage in large processor matrices. Furthermore, the use of communication along diagonals means that it is well-suited for future implementation on the *MASPAR* computer, whose architecture is described in Section 5.1.

The requirement for this method is that the processor matrix size be $N_p \times N_p$.

First, the times of arrival, t_i, are stored in the first row of the processor matrix, $\mu_{0,i}$, $i = 0, \ldots, N_p - 1$. Next, the first differences, $\Delta_{i,i+1}$, are calculated and stored in the first column of the matrix, $\mu_{i+1,0}$, $i = 0, \ldots, N_p - 2$ as shown in Figure 1(a). The next step is to add the contents of the adjacent elements of the first processor column to obtain the $\Delta_{i,i+2}$ since

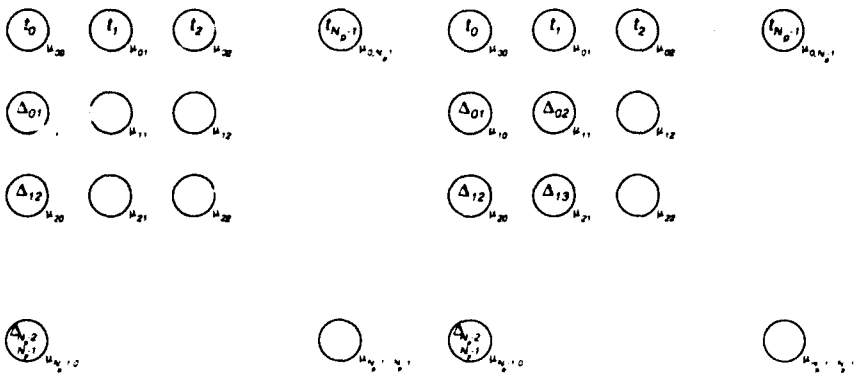
$$\Delta_{i,i+1} + \Delta_{i+1,i+2} = \Delta_{i,i+2}.$$

The results are then stored in the adjacent processor to the right, i.e. the second column. The contents of the processor matrix are then as shown in Figure 1(b). The remaining elements can be computed using the following identity:

$$\Delta_{l,k} = \Delta_{l,k-1} + \Delta_{l+1,k} - \Delta_{l+1,k-1},$$

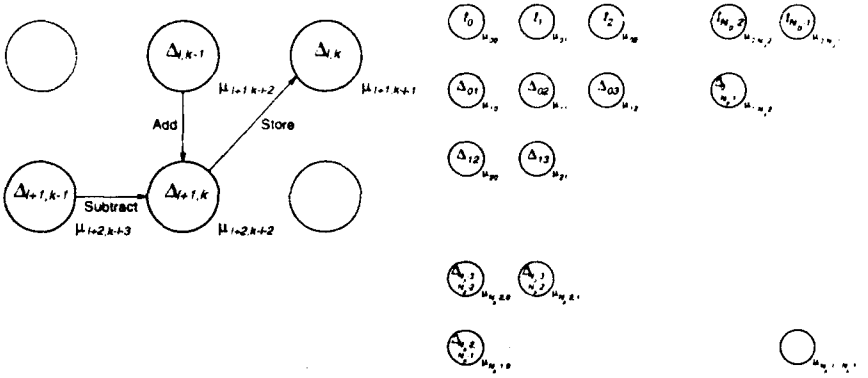
where $0 < l \leqslant k < N_p$ This differencing operation is illustrated in Figure 1(c). This step is repeated until all elements of the Δ matrix have been calculated. The configuration of the matrix at the end of the matrix is shown in Figure 1(d).

The operations which calculate the values for each new column in the matrix are independent of one another, and therefore can be performed in parallel. For each element of the column, the number of operations is fixed. The number of sequential steps that must be performed is therefore determined by the number of columns, N_p. Hence, this method requires $O(N_p)$ steps to complete. The number of processors required is $O\left(N_p^2\right)$, and, because only one variable needs to be stored on each processor, the storage requirements per processor is $O(1)$.



(a) State of the processor matrix after first time differences have been calculated.

(b) State of the processor matrix after second time differences have been calculated.



(c) Method for calculating higher order differences.

(d) Method for calculating higher order differences.

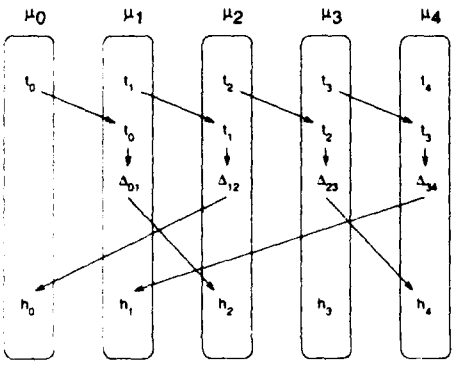
Figure 1 Method 1 for SIMD Architectures

3.2 Method 2

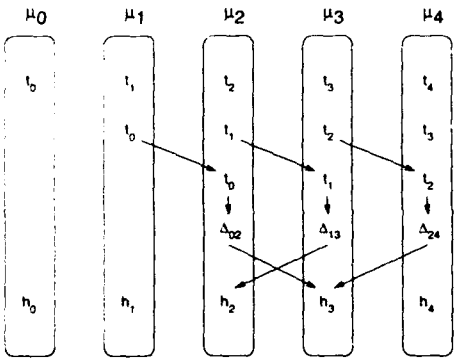
A method is presented here which calculates the TDOA histogram, instead of the TOA matrix. The processors are viewed as a 1-dimensional array rather than as a matrix. The times of arrival t_i are mapped onto the processors μ_i so that each processor holds one time of arrival. Each processor also holds one histogram bin, i.e. h_i is stored on μ_i. This is illustrated in Figure 2(a).

For the first step of the algorithm, each processor passes — "shifts" — a copy of its TOA to its neighbour on the right, i.e. t_i and t_{i-1} are now stored on μ_i. The differences, d_i, of the pairs are now calculated so that the μ_i now also contain $\Delta_{i-1,i}$, $i > 0$. Note that μ_0 becomes inactive since t_{-1} is not recorded. The histogram bin to which the difference belongs is given by the expression

$$p_i = \lfloor d_i / t_b \rfloor . \tag{2}$$



(a) Initiation of the histogram with first order differences using Method 2 for $N_p = 5$.



(b) Updating the histogram with second order differences using Method 2 for $N_p = 5$.

Figure 2 Method 2 for SIMD Architectures

The processor which stores the relevant histogram bin for the difference stored in μ_i is therefore μ_{p_i}. Hence, μ_i sends a signal to μ_{p_i} so that the bin is incremented. However, if $p_i > N_p$ then no processor is responsible for the histogram bin and no processor is signalled. The second order differences are then computed by again passing the copy of the received TOA on to its neighbour so that μ_i, $i > 1$, now holds t_i and t_{i-2}. Both μ_0 and μ_1 are now inactive. Again, the difference of the t's are calculated and the appropriate histogram bin signalled (if in range). This is shown in Figure 2(b).

The action of shifting, differencing and signalling is repeated in that order until all processors are inactive (i.e. all differences have been calculated), or until a stage is reached where all the p_i are out of range. If the p_i are all out of range, then no changes will be made to the computed histogram in current or future iterations. Once one of these criteria has been satisfied, the histogram has been formed.

The operations in each processor are independent of the others, and so $O(N_p)$ sequential steps are required for this method, assuming that all differences are calculated and that the

signalling between processors can be performed entirely in parallel. The storage requirements are $O(1)$ per processor since only one TOA and one histogram bin need to be stored locally on a processor.

When implementing this method on the *MASPAR*, consideration must be made of the fact that the signalling required by this Method cannot be performed entirely in parallel since a processor can only accept one signal at a time. In the worst case, in which a single periodic pulse train is present in the data, each processor will wish to contact the same histogram bin/processor. Hence, the time requirement for signalling is then $O(N_p)$, and the overall time required is $O\left(N_p^2\right)$.

Note that it is to be expected that this method would be sensitive to bin width because the variable number of differences which must be formed until the histogram is complete for a given pulse density (or mean pulse arrival rate). When the bin width is narrow, or the mean pulse rate is low, the number of orders of differences required is small because, after only a few orders, the differences are large enough that none fall within the range of the histogram. A roughly linear relation between time required and histogram bin width is expected because of the roughly linear relation between number of orders of differences required and histogram bin width. Similarly, a linear relation is expected between time required and mean pulse rate.

3.3 Method 3

The method of forming the histogram in parallel discussed above makes extensive use of a global communications system to notify processors when a histogram bin is to be incremented. On the *MASPAR* computer, such global communications are considered a precious resource. Local communication is much to be preferred.

A second problem with the previous method is that there is significant latency in the processors if all differences are to be calculated. In fact, on average, half of the processors are inactive.

To avoid global communication as much as possible, a partial histogram is stored in a complete set of bins on each processor, and merged in an orderly fashion after all differencing has been performed. The partial histograms are denoted $h'_{i,j}$, $0 \leqslant i,j < N_p$, where the first index denotes the processor number on which it is stored, and the second index is the histogram bin within the processor. To maximise processor utilisation, the TOAs are "rotated" through the processor array rather than shifted. By "rotated" it is meant that the data are passed as previously described, except that the last processor in the array also passes its TOA data to the first element in the array, as illustrated in Figure 3. Hence, after the first rotation, μ_i holds t_i and t_{i-1}, for $i > 0$, and μ_0 holds t_0 and t_{N_p-1}.

The method proceeds by first rotating the data. The absolute differences d_i are then formed. Hence, μ_i holds $\Delta_{i-1,i}$, for $i > 0$ and μ_0 holds Δ_{0,N_p-1}. The index into each processor's own private partial histogram is p_i from (2). Thus, h'_{i,p_i} is incremented. The rotation and differencing steps are performed again so that μ_i now holds t_i, t_{i-2} and $\Delta_{i-2,i}$, for $i > 1$, and t_i, t_{i+N_p-2}, and $\Delta_{i,i+N_p-2}$ for $i = 0,1$. The index is calculated and the appropriate partial histogram bin is incremented as before if it is within the range of bins stored.

These steps are repeated until $N_p/2$ rotations have been performed, assuming N_p is an even number. After the final rotation, each processor μ_i contains t_j and $t_{j+N_p/2}$, where $j = i$, for $i < N_p/2$, or $j = i - N_p/2$ otherwise. Hence, μ_i and $\mu_{i+N_p/2}$ contain the same pair of TOAs. For this reason, on the last step, half of the processors are made inactive to avoid duplicate differences. However, for those processors remaining active the final step proceeds as before.

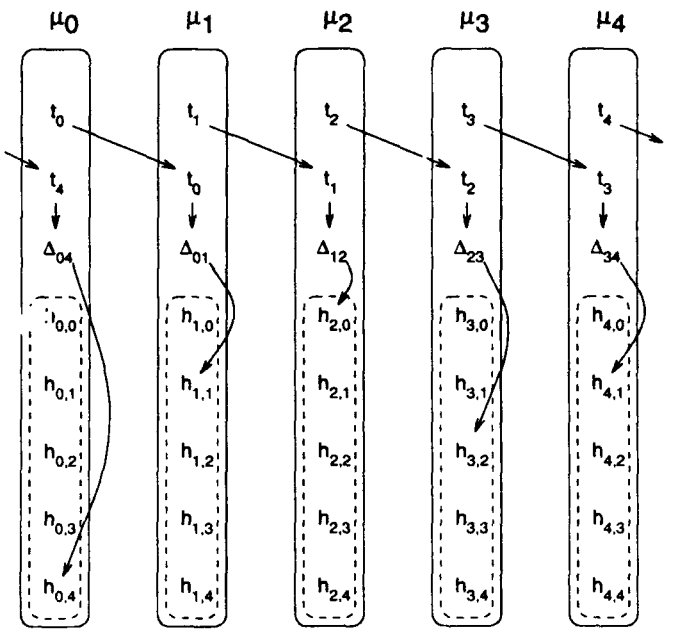


Figure 3 Calculation of the partial histograms of first order differences using Method 3 for $N_p = 5$.

It now remains to merge the partial histograms into a complete histogram. This is performed in a two-dimensional manner, i.e. it makes use of the matrix layout of the processors. More particularly, it makes use of the assumption that the processors are laid out as a *square* matrix and that the number of processors is an integer power of 4. However, the method can be generalised to non-square matrices and to numbers of processors which are not integer powers of 4, although the generalisations are not presented here.

The merging process proceeds by gathering and distributing fewer and fewer partial results further and further afield, until the partial results sum to give the complete histogram. The algorithm for merging proceeds as follows:

i. Let all the partial histogram bins on each processor be declared "current."

ii. Let the distance to neighbours, ϵ_s, equal 1.

iii. Each processor chooses a quarter of its partial bins to remain current in a way determined by its processor number.

iv. The remainder of the bins which were previously current are distributed to the three neighbours who are a distance ϵ_s away in the eastern, southern and south-eastern directions.

v. Simultaneously, each processor's current bins are updated with information received (or fetched) from neighbours who are a distance ϵ_s away in the western, northern and north-western directions.

vi. If the number of current bins per processor equals one, then the merge has finished and the set of current bins from each processor constitutes the complete histogram.

vii. Otherwise, double the distance to neighbours, ϵ_s, and repeat from step iii.

The algorithm is now explained in greater detail. For each step s, $s = 1, \ldots, \log_4 N_p$, the partial histogram on each processor is updated thus

$$h'_{i,j,k,l} \longleftarrow h'_{i,j,k,l} + \begin{cases} h'_{i+\epsilon_s,j,k,l} & \text{when } i \bmod 2\epsilon_s = k \bmod 2\epsilon_s, \\ +h'_{i,j+\epsilon_s,k,l} & j \bmod 2\epsilon_s = l \bmod 2\epsilon_s, \\ +h'_{i+\epsilon_s,j+\epsilon_s,k,l}, & \\ 0, & \text{otherwise,} \end{cases} \tag{3}$$

where $h'_{i,j,k,l} = h'_{(i \bmod N_y)N_x+(j \bmod N_x),kN_x+l}$, for all $0 \leqslant i,k < N_y$, $0 \leqslant j,l < N_x$ and $\epsilon_s = 2^{s-1}$. After all steps have been completed, $h'_{i,i} = h_i$ and the histogram is complete.

For step s of the merging process, each process must fetch $3N_p/4\epsilon_s^2$ data elements over a matrix distance of ϵ_s processors: horizontally, vertically, and diagonally. If a cost of $\tau_f = O(\epsilon_s)$ is assigned to fetching data over a distance ϵ_s (as it is on the *MASPAR*), then the total time requirement for the merging process is

$$\begin{aligned} \tau_m &= \sum_{s=1}^{\log_4 N_p} \frac{3N_p\tau_f}{4\epsilon^2} \\ &= \frac{3N_p}{4} \sum_{s=1}^{\log_4 N_p} \frac{O(2^s)}{4^s} \\ &= \frac{3N_p}{4} \sum_{s=1}^{\log_4 N_p} O(2^{-s}) \\ &= \frac{3N_p}{4} O(1) \\ &= O(N_p). \end{aligned}$$

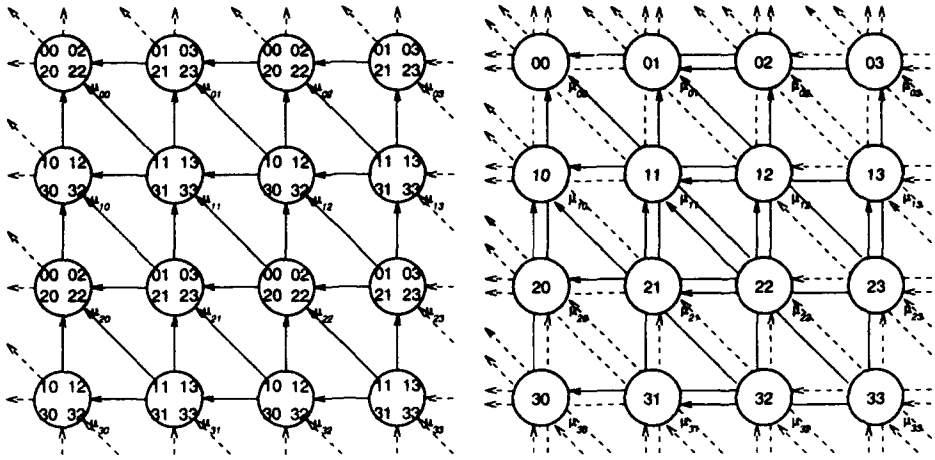
Hence, the time required for merging is not dependent on the form of the data as it is in Method 2, but on the size of the data. This may be an advantage in system design, since the time required to compute the histogram can be predicted with a high degree of accuracy.

An example of the merging algorithm for the case of a small processor matrix ($N_p = 16$, $N_x = N_y = 4$) is shown in Figure 4. The figure illustrates the two merging steps which are required to merge the partial histograms into a complete histogram. The pairs of numbers displayed in each processor circle represent the i,j pairs of the $h'_{i,j,k,l}$ which are to be fetched from other processors. The arrows indicate from which processors the data is being sent. Dashed arrows indicate that the toroidal nature of the matrix is being exploited.

Both the differencing and merging processes require $O(N_p)$ steps to complete and so Method 3 requires $O(N_p)$ steps. The storage requirements per processor are now $O(N_p)$ also since a partial histogram is stored on each processor. However, the linear relation between pulse rates or bin widths and time required is no longer expected as it was for Method 2. Rather, it is to be expected that the algorithm would require a fixed amount of time, depending only on the number of pulses in the buffer, because all loops in the algorithm iterate for a fixed number of times, independent of bin widths and pulse arrival rates.

3.4 Comparison of Resource Requirements

It is possible to compare the methods described above in terms of their usage of important computing resources, such as time, processors and memory. Such a comparison is presented in Table 1, where time, processors and local and global memory have been listed as functions of the number of data points in order notation. Note that the stated requirements for local memory are *per processor*.



(a) First merge. (b) Second and final merge.

Figure 4 Example of the merging algorithm of Method 3 for a 4×4 processor grid.

Table 1 Comparison of resource requirements for SIMD methods.

Method	Time	Processors	Local mem.	Global mem.
Method 1	$O(N_p)$	$O\left(N_p^2\right)$	O(1)	O(1)
Method 2	$O\left(N_p^2\right)$	$O(N_p)$	O(1)	O(1)
Method 3	$O(N_p)$	$O(N_p)$	$O(N_p)$	O(1)

4 HISTOGRAMMING USING THE VECTOR PROCESSOR ARCHITECTURE

The vector processor model is in some ways similar to the SIMD model. A vector processor typically consists of a number of vector "units". Each of these units is roughly analogous to a simple, specialised SIMD processor array. Within the vector unit, a simple hard-coded operation, specific to the unit, is performed on an array or vector of data, either in a pipeline or in parallel or both. Typically, vector units are built for vector addition, subtraction, multiplication and division and operations which cannot make use of the specialised vector units are performed serially according to the usual von Neumann SISD model. The economies of scale which are achieved by using vector units over a standard arithmetic logic unit (*ALU*) are very significant, and this is why they were and are still popular with manufacturers of supercomputers.

The method of performing histogramming here is somewhat naïve. The vectors $v_i, i = 0, \ldots, N_p/2 - 1$ are formed, where v_i is the vector t of (1) rotated i places. Hence

$$v_0 = \left[\begin{array}{cccc} t_0 & t_1 & t_2 & \cdots & t_{N_p-1} \end{array} \right]^T = t$$

$$
\begin{aligned}
v_1 &= \left[\, t_{N_p-1} \quad t_0 \quad t_1 \quad \cdots \quad t_{N_p-2} \,\right]^T \\
v_2 &= \left[\, t_{N_p-2} \quad t_{N_p-1} \quad t_0 \quad \cdots \quad t_{N_p-3} \,\right]^T \\
&\vdots
\end{aligned}
$$

The algorithm proceeds from step $s = 1, \ldots, N_p/2 - 1$. At each step s, the operation

$$a_s = |v_s - v_0|$$

is performed, where a_s is a vector of absolute time differences and in this case $|\cdot|$ is the absolute value of each element of the vector. This operation can be efficiently performed in a vector difference unit. The method of rotation and absolute differencing is analogous to that performed in Method 3 in Section 3.3. It results in all of the Δ_{ij} being calculated except for the $\Delta_{i,i+N_p/2}$. As with Method 3 above, a final special step, $s = N_p/2$, must be performed in which two vectors of rank $N_p/2$ are formed and differenced so that

$$a_{N_p/2} = |\mathrm{row}(t, N_p/2, N_p/2) - \mathrm{row}(t, 0, N_p/2)|,$$

where $\mathrm{row}(x, i, j)$ is the vector formed from j rows of vector x starting from row i.

At the end of each step, the elements of the vector a_s are sorted into the appropriate histogram bins. The division which is required to calculate the appropriate bins for each element can make use of a vector division unit. On the Fujitsu VP-2200 (which was used to implement this algorithm), the vector difference and division units were the only two used. All other operations were performed in the traditional serial manner. An illustration of the implementation of the algorithm on a vector processor with $N_p = 5$ is shown in Figure 5.

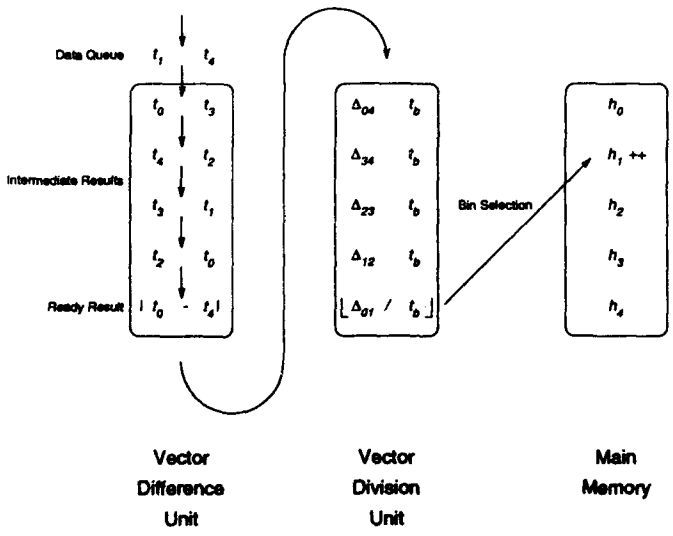


Figure 5 Implementation of parallel histogramming on a vector processor.

The time required for vector processing in the way described above requires $\mathrm{O}\left(N_p^2\right)$ computational steps and the storage requirements are $\mathrm{O}(N_p) + \mathrm{O}(N_H)$ (since it is not necessary for this algorithm to have $N_p = N_H$).

5 NUMERICAL SIMULATIONS

Numerical simulations were performed on several types of computers using the algorithms described previously. A most important feature of deinterleaving from an operational perspective is the time required for deinterleaving a given number of points. Throughout the following discussions, sets of data points containing the times of arrival of 4096 pulses in double precision real numbers were histogrammed. This number of pulses was chosen because it matched the number of processors on the *MASPAR* computer. The time required to perform the ***histogram only*** was measured. Hence, the measurements take no account of the time required to read the TOA's off disk or to display or otherwise process the results of the histogram subsequently. While these times may indeed be critical to the performance of a deinterleaving system, it is outside the scope of this report.

5.1 The *MASPAR* Architecture

The *MASPAR* MP-1 consists of two major functional parts: the Array Control Unit (ACU), and the Processor Elements (PE's) [3,5]. The ACU has at its core a conventional SISD RISC (Reduced Instruction Set Computer) microprocessor which coordinates the actions of the PE's. The ACU can communicate both with the PE's or with its host computer (in this case a DECstation 5000) via VME bus. The PE's themselves can communicate with the ACU and with each other. Two methods exist for communication between PE's, and these correspond with the methods described in Section 3. The first is a nearest neighbour mechanism, through which each PE can communicate with its nearest neighbour around the four major compass points and around the four minor compass points. This mechanism is called the "Xnet." It can also be used for communication between processors over a specified distance along the major and minor compass points. The second mechanism is a means by which any PE can communicate with any other PE, rather like a telephone network. This mechanism is known as the "Global Router." However, the *MASPAR* PE's are grouped into blocks of 16 and each block has just one router line. Hence, each PE within the block must compete for use of the "party line." Additionally, this is a slower method of communication than the Xnet, and so Xnet is to be preferred where convenient. A simplified system diagram of the *MASPAR* is shown in Figure 6.

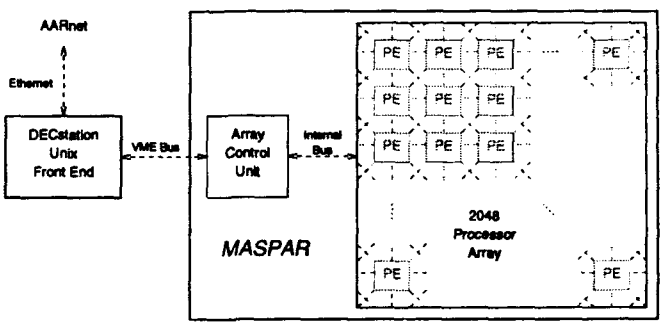


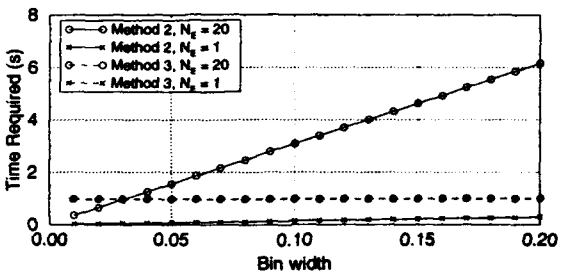
Figure 6 System overview of the *MASPAR* MP-1 computer.

5.2 Implementation on the *MASPAR*

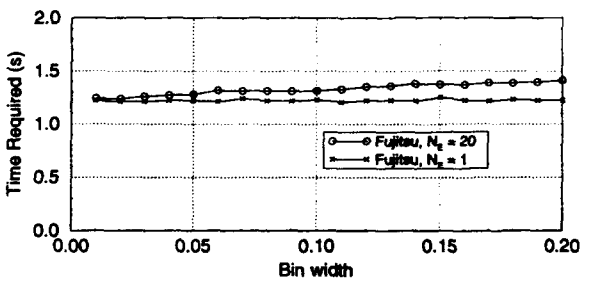
Methods 2 & 3 of the SIMD algorithms were implemented on a *MASPAR* computer in MPL (*MASPAR* Programming Language) [4]. The *MASPAR* which was used for the numerical simulations has 4096 individual processors ($N_p = 4096$), laid out as a 64×64 two-dimensional grid. Method 2 makes use of both the Xnet for the data shifting and the global router for incrementing histogram bins. Method 3 makes use of the Xnet only, both for data rotation and the final histogram merge. Listings of the MPL code are contained in Appendix Sections I.1–I.3.

The two methods were trialled for two pulse data sets, and for various histogram bin widths. Each pulse data set contains 4096 pulses. The first data set contained 20 separate periodic emitters with PRI's ranging from 10.1 to 19.3 arbitrary time units, with an average PRI of 14.91 units and a mean pulse rate of one pulse per 0.7161 units. The second data set contains a single periodic emitter with a PRI of 15 units. Histogram bin widths were varied between 0.01 units and 0.2 units

The times required to compute the TDOA histograms for each method were measured, and these are plotted in Figure 7(a). From the plot, it can be seen that Method 2 is sensitive



(a) Computation time vs. histogram bin width for *MASPAR* simulations.



(b) Computation time vs. histogram bin width for Fujitsu simulations.

Figure 7 Results of numerical simulations.

to both bin width and mean pulse rate, whereas Method 3 is completely insensitive to both of these factors, as predicted in Sections Section 3.2 and Section 3.3. That is, Method 2 is sensitive to bin width and mean pulse rate because its main loop terminates as soon as the

minimum difference at that order is larger than the maximum PRI being recorded in the histogram. Method 3 is insensitive to both factors since the complete set of differences is always calculated. It is also found that the histogram merging operation offers substantial time savings over contacting processors individually after each difference operation (as used in Method 2) if the full set of differences is calculated. The time required for histogram merging in Method 3 is 0.2943 seconds whereas Method 2 requires 0.0054 seconds for interprocessor communication after each round of differencing. Hence if more than about the 54th order of differences is required, the communication time required for Method 3 will be less than that required by Method 2.

5.3 The Fujitsu VP–2200 Architecture

The Fujitsu VP–2200 is a traditional vector processor. It consists of several specialised processing units, along with a general von Neumann-type SISD processor.

5.4 Implementation on the Fujitsu VP–2200 Architecture

The method as outlined in Section 4 was implemented in Fortran on the Fujitsu VP–2200. It was compiled using all vectorising options turned on to maximise the use of the vector units. A listing of the Fortran subroutine which was used to calculate the histogram is presented in Section I.4.

The method was tested for both data sets described in Section 5.2. As for the *MASPAR*, the times required for histogramming were measured and they are graphed in Figure 7(b). Again, there is a linear relation between binwidth and time required, and the relationship is also dependent on mean pulse repetition frequency. The dependency on both binwidth and mean PRF is much reduced, since all differences are calculated, regardless of the form of the data. The dependency occurs in the step where differences are sorted into bins. If the histogram array index is out of bounds, no increment can be performed. In situations where the mean PRF is low compared with the binwidth, the indices will all be out of bounds after a few iterations, so the histogram will be formed in slightly less time.

5.5 Comparisons Between Implementations

A comparison between architectures and algorithms can be made by selecting a certain set of parameters, running the algorithms on each of the architectures with these parameters, and measuring the time required to compute the histogram. This was done for the *MASPAR* MP–1 for Methods 2 and 3, and on the Fujitsu VP–2200, Hewlett-Packard Series 9000 Models 720 and 730 workstation, and on the Sun SPARC 2 workstation using the Fortran algorithm. The parameters used in the tests were a bin width of 0.1 units and the first data set (consisting of 4096 pulses from 20 periodic emitters with mean PRI of 0.7161 units).

The following table summarises the results obtained. The results show that the best algorithm and architecture in this case is the *MASPAR* using Method 3. The rate of calculation corresponds to approximately 4000 pulses per second.

Table 2 Comparisons between architectures and algorithms.

Architecture	Algorithm/Language	Time (s)
MASPAR MP-1	Method 2/MPL	3.09
	Method 3/MPL	0.98
Fujitsu VP-2200	Vectorised Fortran	1.31
HP 720	Optimised Fortran	7.20
HP 730	Optimised Fortran	5.44
Sun SPARC 10	Optimised Fortran	11.70
Sun SPARC 2	Optimised Fortran	33.77

6 SUGGESTED IMPROVEMENTS

A number of changes could be made to the algorithms to improve the speed of calculation of the histogram. Improvements which would benefit all algorithms and architectures are:

i. use of fixed point (integer) rather than double precision floating point;

ii. expressing all times of arrival in binwidth units. That is, before any differencing takes place, all of the TOAs would be divided by the binwidth. If this improvement were applied, the integer part of the difference would become the index of the histogram bin. Also, the number of divisions required would now be $O(N_p)$ rather than $O\left(N_p^2\right)$, although the total number of operations would still be $O\left(N_p^2\right)$.

The adoption of these improvements on an HP 730 gives a measured total time of 1.19 seconds, when tested under the conditions used in Section 5.5. If the greater than four times speed up can be transferred to the *MASPAR* (a reasonable assumption), then a histogram calculation rate of nearly 20,000 pulses per second may be possible.

Specific to the *MASPAR*, the following changes might further increase the processing rate:

i. storage of the entire TOA records in each processor element, rather than merely storing two TOAs, which would reduce some of the inter-PE communication overhead;

ii. pipelining the histogram calculations. For instance, the PE array could be divided into four groups. Each group would work at a different stage in the calculation process.

Note that, because the histogram is an $O\left(N_p^2\right)$ process ($O(N_p)$ for the *MASPAR*), reducing the number of pulses to be histogrammed by a factor of two results in a speed increase of up to four times. Further, it may not be necessary to calculate all the orders of difference. Reducing the number of orders of difference to a fixed number, potentially reduces the order of the process from $O\left(N_p^2\right)$ to $O(N_p)$. For the examples used in this report in which 4096 pulses have been used, then limiting the calculation of the histogram up to the 200th order of difference may yield a speed improvement of around 10 times. However, the more crowded the pulse environment, the more orders will be required to identify a sufficient number of harmonics for detection.

7 FUTURE DIRECTIONS

Many improvements have been discussed which have not yet been rigorously implemented and tested. It is planned that some or all of these will be tested. Further, there are some architectures which have not been examined and which deserve attention. Notable amongst these are the hypercube SIMD architecture employed by the Connection Machine, and MIMD architectures such as transputer arrays.

The related question of how to perform detection of PRI's from the histograms once they have been computed is an important topic for further research. In particular, the use of neural networks in detection, and how to present TOA data to a neural network needs to be investigated.

It is hoped that Method 1, discussed in Section 3.1, being a nearest neighbour algorithm, will be suitable for ready implementation on a hypercube architecture, and with further work, may be useful in neural network investigations.

8 CONCLUSIONS

Four methods have been presented for calculating the time of arrival histogram for a pulse data set. Two architectures have been considered: a SIMD parallel processor (the *MASPAR* MP-1) and a vector processor (the Fujitsu VP-2200). Three of the algorithms and each of the architectures were tested under various conditions. Comparisons were made between the described algorithms and architectures and with more well-known architectures such as Hewlett-Packard and Sun workstations. One algorithm (Method 1) was considered unsuitable for testing and comparison because it does not calculate a histogram.

It was found that the *MASPAR* was able to calculate the histogram fastest. The speed of the algorithms on all architectures was limited by the use of double precision arithmetic and the large number of pulses used in the processing. Given these restrictions, and given also that no account of the time required for input and output of the data, a maximum processing rate of about 4000 pulses per second was achieved.

Several improvements were suggested. The incorporation of these improvements into algorithms on the *MASPAR* may lead to a calculation rate for the full histogram of around 20,000 pulses per second. With the recent release of the new, faster *MASPAR* MP-2, even this number could well be exceeded. However, this has yet to be confirmed.

9 ACKNOWLEDGEMENTS

The authors are grateful to the Center for Information Technology Research (CiTR), University of Queensland, and the Australian Nuclear Science and Technology Organisation (ANSTO), Lucas Heights, for the generous donation of time on their *MASPAR* and Fujitsu computers, respectively.

THIS IS A BLANK PAGE

REFERENCES

1. Vaughan Clarkson, Jane Perkins, and Iven Mareels. On the novel application of number theoretic methods to radar detection. *Proceedings of the International Conference on Signal Processing Applications and Technology*, October 1993.

2. H. K. Mardia. New techniques for the deinterleaving of repetitive sequences. *IEE Proceedings-F*, 136(4):149–154, August 1989.

3. MasPar Computer Corporation, Sunnyvale, California. *Data-Parallel Programming Guide*, 1991.

4. MasPar Computer Corporation, Sunnyvale, California. *MasPar MP-1 MPL Programming Manuals*, July 1991. Part Number 9300-9005-00 Rev A2.

5. MasPar Computer Corporation, Sunnyvale, California. *MasPar MP-1 System Overview & MPPE Manuals*, July 1991. Part Number 9300-9001-00 Rev A5.

6. D. J. Milojević and B. M. Popović. Improved algorithm for the deinterleaving of radar pulses. *IEE Proceedings-F*, 139(1):98–104, February 1992.

7. J. A. V. Rogers. ESM processor system for high pulse density radar environments. *IEE Proceedings*, 132, Pt. F(7):621–625, December 1985.

8. Ralph O. Schmidt. On separating interleaved pulse trains. *IEEE Transactions on Aerospace and Electronic Systems*, pages 162–166, January 1974.

9. Richard G. Wiley. *Electronic Intelligence: The Analysis of Radar Signals*, pages 160–163. Artech House, Norwood, Massachusetts, 1982.

THIS IS A BLANK PAGE

APPENDIX I

PROGRAM LISTINGS

The following are listings of the code used in generating the benchmarks in Section 5 of this report. Only the subroutines which generate the histogram are reproduced here: the main body of the programs which include reading in the TOA data from files and decoding command line options, and the code for timing and profiling, has been omitted.

Sections I.1-I.3 are listings of the MPL code used on the *MASPAR*. They consist of two versions of `hist.m` as well as a header file, `vlc_mpl.h`.

Section I.4 is a listing of the Fortran subroutine `calcHist` used both on the Fujitsu VP-2200 vector processor and for comparison on the Hewlett-Packard 720 and 730 and Sun SPARC 2 workstations.

I.1 SIMD Method 2 — `hist.m`

```
#include <mpl.h>
#include <math.h>
#include <sendwith.h>
#include "vlc_mpl.h"

plural bool
Histogram(toa, PRIspacing)
  register plural double  toa;
  register double         PRIspacing;
{
  register int            order = 0;
  register plural bool    result;
  register plural procno  hist_bin;
  register plural procno  my_hist = 0, to_send;
  register plural double  next_toa, pass_toa, diff;
  register double         timeRange;
  register plural bool    destInRange = TRUE, activeSet;

  /* Pre: the toa are in ascending order with iproc */

  timeRange = proc[nproc-1].toa - proc[0].toa;
  pass_toa = toa;
  hist_bin = iproc;
  to_send = 0;
  while ((++order < nproc) && (iproc >= order) && destInRange) {
    if (ixproc > 0)
      next_toa = xnetW[1].pass_toa;
    else
      next_toa = xnetNW[1].pass_toa;
    pass_toa = next_toa;
    diff = toa - next_toa;
    hist_bin = (plural int) p_floor(diff / PRIspacing);
    destInRange = FALSE;
```

```
31      to_send = 0;
32      if (hist_bin < nproc) {
33        destInRange = TRUE;
34        to_send = 1;
35      }
36      all {
37        my_hist += sendwithAdd16u(to_send, hist_bin);
38        hist_bin = iproc;
39        to_send = 0;
40      }
41    }
42    return(my_hist);
43  }
```

I.2 SIMD Method 2 — vlc_mpl.h

```
1  /* Useful MPL declarations.

2  Written by Vaughan Clarkson, April, 1991.
3  Copyright (c) 1991.
4  */

5  /* Definitions for boolean variables.  Should be an enumerated type but
6     presently MPPE can't cope with enums
7  */
8  #define FALSE 0
9  #define TRUE (!FALSE)

10 /* This awkward construction overcomes the limitation of the natural
11    MPL while statement by removing the restriction that the active set
12    is non-increasing
13 */
14 #define FULLwhile(pvar) while(reduceOr8u(pvar)) if (pvar)

15 /* Wrap around xnet so that PE are arranged linearly */
16 #define xnetLEFT(pvar) (ixproc > 0 ? (xnetW[1].pvar) : (xnetNW[1].pvar))
17 #define xnetRIGHT(pvar) (ixproc < nxproc-1 ? (xnetE[1].pvar) : (xnetSE[1].pvar))

18 typedef unsigned char bool;
19 typedef unsigned short procno;
```

I.3 SIMD Method 3 — hist.m

```
1  #include <mpl.h>
2  #include <math.h>

3  #define NULL 0
```

```
 4  plural int histogram(/* register plural double */ toa,
 5  /* register double */ binWidth)
 6    register plural double toa;
 7    register double binWidth;
 8  {
 9    plural unsigned short myHist[4096];
10    int order;
11    plural double nextTOA, passTOA, diff;
12    plural int histBin;
13    int foundationMask, upperMask, lowerMask, distance;
14    int numToSend, upper, lower;
15    plural int foundation, myIndex;

16    passTOA = toa;
17    for (order = 0; order < nproc / 2 - 1; order++) {
18      if (ixproc > 0)
19        nextTOA = xnetW[1].passTOA;
20      else
21        nextTOA = xnetNW[1].passTOA;
22      passTOA = nextTOA;
23      diff = p_fabs(toa - nextTOA);
24      histBin = (plural int) p_floor(diff / binWidth);
25      if (histBin < nproc)
26        myHist[histBin]++;
27    }
28    /* And again for the final iteration, though now with just half the PEs */
29    if (iproc < nproc / 2) {
30      if (ixproc > 0)
31        nextTOA = xnetW[1].passTOA;
32      else
33        nextTOA = xnetNW[1].passTOA;
34      passTOA = nextTOA;
35      diff = p_fabs(toa - nextTOA);
36      histBin = (plural int) p_floor(diff / binWidth);
37      if (histBin < nproc)
38        myHist[histBin]++;
39    }

40    /* Now redistribute the histograms */
41    /* Assumptions: nxproc == nyproc and nproc is an integer power of 4 */
42    /* Can be generalised. */
43    foundationMask = nxproc | 1;
44    upperMask = nxproc << 1;
45    lowerMask = 2;
46    distance = 1;
47    for (numToSend = nproc >> 2; numToSend > 0; numToSend >>= 2) {
48      foundation = iproc & foundationMask;
49      for (upper = 0; upper < nproc; upper += upperMask)
50        for (lower = 0; lower < nxproc; lower += lowerMask) {
51  plural unsigned short dummyE, dummyS, dummySE;
52  plural int indexE, indexS, indexSE;
```

```
53 myIndex = foundation | upper | lower;
54 indexE = xnetW[distance].myIndex;
55 indexS = xnetN[distance].myIndex;
56 indexSE = xnetNW[distance].myIndex;
57 dummyE = xnetE[distance].myHist[indexE];
58 dummyS = xnetS[distance].myHist[indexS];
59 dummySE = xnetSE[distance].myHist[indexSE];
60 myHist[myIndex] += dummyE + dummyS + dummySE;
61       }
62     foundationMask |= foundationMask << 1;
63     upperMask <<= 1;
64     lowerMask <<= 1;
65     distance <<= 1;
66   }
67   return(myHist[iproc]);
68 }
```

I.4 Vector/SISD Fortran code

```
 1       subroutine calcHist(toa, hist, copy, diff, index)
 2       real*8 toa(*), copy(*), diff(*)
 3       integer hist(*), index(*)
 4 c
 5 c     Calculate the histogram from the times of arrival (TOAs) in toa
 6 c     and place the results in hist.  The arrays copy, diff and index
 7 c     are scratch arrays.  All arrays except hist must be at least of
 8 c     dimension numtoa, and hist must have a minimum dimension of numhist.
 9 c
10       common /histparams/ numtoa, numhist, binwidth
11       integer numtoa, numhist
12       real*8 binwidth

13       integer i, j

14 c     Assume even number of toas
15       do 200 i = 1, numtoa / 2 - 1
16          do 300 j = 1, numtoa
17             if (j .le. i) then
18                copy(j) = toa(numtoa - i + j)
19             else
20                copy(j) = toa(j - i)
21             endif
22             diff(j) = abs(toa(j) - copy(j))
23             index(j) = int(diff(j) / binwidth) + 1
24             if (index(j) .le. numhist) then
25                hist(index(j)) = hist(index(j)) + 1
26             endif
27  300     continue
28  200  continue
29       do 400 j = 1, numtoa / 2
```

```
30          diff(j) = abs(toa(j + numtoa / 2) - toa(j))
31          index(j) = int(diff(j) / binwidth) + 1
32          if (index(j) .le. numhist) then
33             hist(index(j)) = hist(index(j)) + 1
34          endif
35   400  continue
36        end
```

THIS IS A BLANK PAGE

DISTRIBUTION

	No of Copies
Defence Science and Technology Organisation	
Chief Defence Scientist)	
Central Office Executive)	1 shared copy
Counsellor, Defence Science, London	Cont Sht
Counsellor, Defence Science, Washington	Cont Sht
Scientific Adviser POLCOM	1
Senior Defence Scientific Adviser	1
Assistant Secretary Scientific Analysis	1
Navy Office	
Navy Scientific Adviser	1
Air Office	
Air Force Scientific Adviser	1
Army Office	
Scientific Adviser, Army	1
Electronics Research Laboratory	
Director	1
Chief, Communications Division	Cont Sht
Chief, Information Technology Division	Cont Sht
Chief, Guided Weapons Division	Cont Sht
Chief, Electronic Warfare Division	1
Research Leader , Signal and Information Processing	1
Head, Signal and Information Processing	1
Mr V. Clarkson (Co-author)	1
Dr P.S. Ray(Co-author)	3
Head, Electronic Support Measures Systems	1
Dr G.P. Noone	1
Dr S.D. Elton	1
Dr D. Sweet	1
Cooperative Research Centre for Robust and Adaptive Systems	
Dr I.M.Y. Mareels, Department of Systems Engineering, Australian National University, ACT 2000	1
Prof D.A. Gray CSSIP, SPRI Building, Warrendi Road, Technology Park, The Levels 5095	1
Dr J.E. Perkins Building 71 Labs DSTO Salisbury	1
M. Balin Building 71 Labs DSTO Salisbury	1
B.J. Slocumb Electronic Systems Laboratory, Georgia Tech Research Institute, Atlanta, Georgia 30332, USA	1

	No of Copies
Libraries and Information Services	
Australian Government Publishing Service	1
Defence Central Library, Technical Reports Centre	1
Manager, Document Exchange Centre, (for retention)	1
National Technical Information Service, United States	2
Defence Research Information Centre, United Kingdom	2
Director Scientific Information Services, Canada	1
Ministry of Defence, New Zealand	1
National Library of Australia	1
Defence Science and Technology Organisation Salisbury, Research Library	2
Library Defence Signals Directorate, Melbourne	1
British Library Document Supply Centre	1
Spares	
Defence Science and Technology Organisation Salisbury, Research Library	6

Department of Defence

DOCUMENT CONTROL DATA SHEET

1. Page Classification	
UNCLASSIFIED	
2. Privacy Marking/Caveat (of document)	
N/A	

3a. AR Number	3b. Laboratory Number	3c. Type of Report	4. Task Number
AR-008-135	ERL-0692-RR	RESEARCH REPORT	ADF 91/136

5. Document Date	6. Cost Code	7. Security Classification	8. No. of Pages	32
October 1993	819908	* U (Document) U (Title) U (Abstract)	9. No. of Refs.	9

10. Title

PARALLEL PROCESSING FOR DIFFERENCING AND HISTOGRAMMING

S (Secret) C (Confi) R (Rest) U (Unclass)

* For UNCLASSIFIED docs with a secondary distribution LIMITATION, use (L) in document box.

11. Author(s)

Vaughan Clarkson and Pinaki S. Ray

12. Downgrading/Delimiting Instructions

N/A

13a. Corporate Author and Address

Electronics Research Laboratory
PO Box 1500, Salisbury SA 5108

13b. Task Sponsor

ADF

14. Officer/Position responsible for

Security: N/A

Downgrading: N/A

Approval for Release: DERL

15. Secondary Release Statement of this Document

APPROVED FOR PUBLIC RELEASE

16a. Deliberate Announcement

No limitation

16b. Casual Announcement (for citation in other documents)

[√] No Limitation [ ] Ref. by Author , Doc No. and date only.

17. DEFTEST Descriptors	18. DISCAT Subject Codes
Parallel processing, Histograms, Time of arrival (signals)	1711

19. Abstract

Algorithms which take advantage of massively parallel and vector architectures are formulated for the calculation of the differences of times-of-arrival derived from a sequence of time samples. The application which generated the work is the classification of pulse trains which are detected by an electronic support measures receiver.

Four algorithms are proposed, of which three have been implemented and tested on several machines. The results obtained through testing are compared with each other and with results for more conventional architectures. It is found that one of the machines, the massively parallel *MASPAR* MP-1 computer, is able to perform histogramming fastest through exploitation of its data-parallel architecture. Several suggestions are made which could significantly improve performance on all architectures.