UNLIMITED

RSRE
MEMORANDUM No. 4213

ROYAL SIGNALS & RADAR ESTABLISHMENT

THE APPLICATION OF A DISTRIBUTED ARRAY PROCESSOR (DAP) TO LINEAR ASSIGNMENT PROBLEMS IN RADAR TRACKING

Author: A J Stanley, P Simpson & J B G Roberts

PROCUREMENT EXECUTIVE,
MINISTRY OF DEFENCE,
R S R E MALVERN,
WORCS.

R S R E MEMORANDUM No. 4213

ROYAL SIGNALS AND RADAR ESTABLISHMENT

Memorandum 4213

TITLE: THE APPLICATION OF A DISTRIBUTED ARRAY PROCESSOR (DAP) TO LINEAR ASSIGNMENT PROBLEMS IN RADAR TRACKING

AUTHORS: A J Stanley*, P Simpson and J B G Roberts

DATE: July 1988

SUMMARY

A Distributed Array Processor (DAP) is an SIMD parallel processing machine composed of 1024 one-bit processing elements (PEs). This Memorandum examines the application and detailed performance of this machine to the linear assignment problem with data arrays up to 256x256 in size. The linear assignment problem is used in ESM, radar tracking, and other fields where it is necessary to assign data from two or more classes to each other. Since the assignment problem is solved by a computationally intensive algorithm a comparison is made between the DAP and a serial machine, a VAX 8600, to assess the speed gains obtained from the DAP by executing instructions in parallel. The results show that the DAP is far faster at solving this problem than the VAX by up to two orders of magnitude.

*Currently at Sheffield City Polytechnic.

CONTENTS

1. Introduction

Linear assignment is the problem of taking values from two or more classes and assigning these values to each other according to a given cost function. The problem arises in a number of practical military applications such as radar tracking where, for example, it may be required to assign radar plots from a radar scan with earlier formed tracks in a single sensor system, or assign radar tracks from multiple radar sensors. Its detailed application to radar tracking will be explained in section 3.

The DAP Fortran code for the linear assignment problem, was based on a library routine of the Hungarian Algorithm[1] written by Mr. J. Yadegar and Dr. A. Frieze, Department of Computer Science and Statistics, Queen Mary College, London. This code was written for a problem which 'fits' the physical architecture of the DAP, i.e. a 64*64 matrix for the original DAP, a 32*32 matrix in the case of MilDAP.

The objective was to evaluate the library routine supplied by QMC by obtaining timings for matrices of sizes up to 32*32 (the physical size of the DAP available), and to generalise it for larger matrices and assess the DAP's performance. These timings were to be compared with those from a sequential computer running the assignment problem.

2. Introduction to the Distributed Array Processor (DAP)

The DAP is a computer which was designed to provide parallel processing capabilities. It has a matrix of 32*32 one bit processing elements (PE's), each having 8K bits of local store, giving a total of 1M byte. It differs from a serial processor in that it can perform the same operation on many items of data in parallel. The simplicity of the PE's provides the user with flexibility. A wide range of operational word lengths can be permitted, allowing speed to be traded directly against arithmetic precision. The DAP is attached to a single-user ICL PERQ which acts as the host. This is responsible for all program development (i.e. compilation, assembly, etc), input/output, and initiates and controls the execution of any DAP program. The host also has a DAP simulator, allowing precise timings of segments of DAP Fortran code, although at a real time speed 31000 times slower than the DAP hardware.

2.1 DAP Fortran

The linear assignment algorithm was written in DAP Fortran, a version of Fortran designed to take advantage of the parallel processing capabilities of the DAP. The most important feature of DAP Fortran is the ability to manipulate matrices and vectors. Matrices and vectors have their first dimensions constrained to 32*32 and 32 respectively, to fit the DAP's hardware. Problems requiring vectors or matrices larger than 32 or 32*32 must be split into a number of vectors or 32*32 matrices.

DAP Fortran differs from Fortran in two main ways:

1. DAP Fortran allows the user to express parallel execution of either an entire vector or matrix, or a subset of the component values of a vector or matrix. A number of indexing techniques are provided for selecting such subsets.
2. DAP Fortran does not have any input/output facilities, so all I/O must be performed by the host program. Data has to be transferred explicitly between the host and DAP programs. Such data must be held in COMMON blocks in both the host and DAP programs.

2.2 Examples of DAP Fortran

Each node of a 32*32 grid has a value that represents the height above sea level of land at that point. The mean height above sea level is calculated in both DAP Fortran and Fortran to show the reduction in code complexity.

Fortran Code

```
      real height(32,32),mean,sum

      sum=0.0
      do 10 j=1,32
      do 10 i=1,32
      sum=sum+height(i,j)
10    continue
      mean=sum/1024
```

DAP Fortran Code

```
      real height(,),mean

      mean=SUM(height)/1024
```

SUM is a built-in DAP Fortran function that, in this example, takes a matrix as argument and returns the sum of all the components of 'height'. DAP Fortran has many other built-in functions that perform parallel operations on vectors and matrices.

An important feature of DAP Fortran is the facility to perform operations only on certain selected components of a vector or matrix. The different indexing techniques are too numerous to detail. However, an example is given below using a logical matrix to index components in an integer matrix.
e.g.
LM is a logical matrix, M1 and M2 are integer matrices.

M1=

1	1	1	1
2	2	2	2
3	3	3	3
4	4	4	4

M2=

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

LM=

T	F	T	T
F	T	T	F
F	F	T	F
T	F	F	F

The result of M1(LM)=M2 is:

1. LM and M2 are unchanged.
2. M1=

1	1	3	4
2	6	7	2
3	3	11	3
13	4	4	4

3. Discussion of the Practical Uses of the Linear Assignment Problem

The linear assignment problem is not solely a theoretical exercise but has a number of practical applications in which it is necessary to make an optimum decision (in the sense of maximising the joint probability) about making associations between objects drawn from two or more classes. In particular, at RSRE linear assignment is of use in plot-track correlation which is used in radar systems.

3.1 Criterion for Plot-Track Correlation[2]

Consider a scanning radar sensor which makes estimates of the (x,y) positions (and possibly velocity, altitude as well) of several targets at each scan. Each parameter estimated is subject to error and the criterion of maximum joint probability is used to determine the optimum associations between the targets detected on each new scan and the target tracks built up from earlier scans. Figure 1 shows a track(e.g. aircraft), T, with its forecast position F deduced from the existing track parameters.

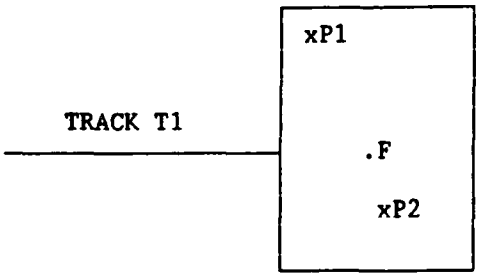


FIGURE 1

A plot from the next scan of the radar should lie in the vicinity of F. The square box around F represents a gate in of which plots from the target are expected to lie. The purpose of this gate is to reduce the number of plots to be considered for association with a particular forecast. The gate should be large enough to include any plots that could possibly be considered as candidates, due allowance being made for noise on the plots, errors in the forecast position, and for target manoeuvre.

Two plots, P_1 and P_2, lie inside the gate. A nearest neighbour correlation algorithm would assign P_2 to F, but if the true situation were as shown in Figure 2, in which track T_2 is in the vicinity of the first track, T_1, then this algorithm would incorrectly assign P_2 to F_1 and P_1 to F_2.

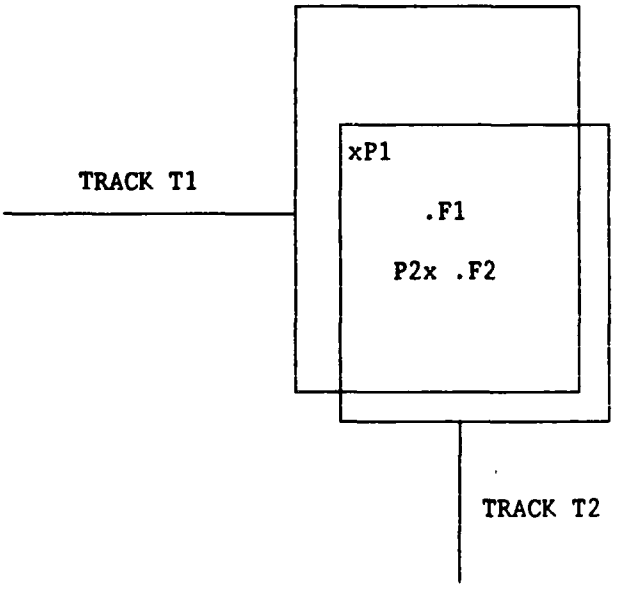


FIGURE 2

Hence, an important principle in plot-track correlation is not to assign nearest neighbour track but, wherever there is uncertainty, to consider all

plots and tracks together.

In the example of Figure 2, what is required is the calculation of four probabilities, namely:

P_{11}	the	probability	that	P_1	associates	with	T_1
P_{12}	"	"	"	P_1	"	"	T_2
P_{21}	"	"	"	P_2	"	"	T_1
P_{22}				P_2			T_2

The joint probability that P_1 associates with T_1 and P_2 associates with T_2 is given by the product $P_{11}P_{22}$. Similarly, the probability that P_1 associates with T_2 and P_2 with T_1 is given by $P_{12}P_{21}$. In this case the better plot to track assignment is the one whose joint probability is the larger.

3.2. The transformation to linear programming[2]

Consider a more general case with three plots P_1,P_2,P_3, and three tracks T_1,T_2 and T_3. The individual probabilities can be written in matrix form:

	T_1	T_2	T_3
P_1	P_{11}	P_{12}	P_{13}
P_2	P_{21}	P_{22}	P_{23}
P_3	P_{31}	P_{32}	P_{33}

There are six different ways of assigning plots to tracks. Six joint probabilities could therefore be calculated, each one of which is the product of these factors. The objection to this method is the need to calculate joint probabilities for all plot to track combinations in order to find the maximum. This difficulty can be removed by transforming the problem from one which requires products of the elements of a matrix to one which requires sum of elements.

Let Q_{ij} be defined by the relation

$$Q_{ij} = -\log P_{ij}$$

where P_{ij} is the probability that the i'th plot associates with the j'th track.

If we write the matrix of Q_{ij} values for the above case of three plots and three tracks we have:

	T_1	T_2	T_3
P_1	Q_{11}	Q_{12}	Q_{13}
P_2	Q_{21}	Q_{22}	Q_{23}
P_3	Q_{31}	Q_{32}	Q_{33}

The criterion for the assignment of plots to tracks changes to become the minimum of the sum of elements of this matrix, one and only one element being taken from each row and column. This is the linear assignment problem.

4. Implementation

4.1. DAP Implementation for Linear Assignment problems up to 32*32

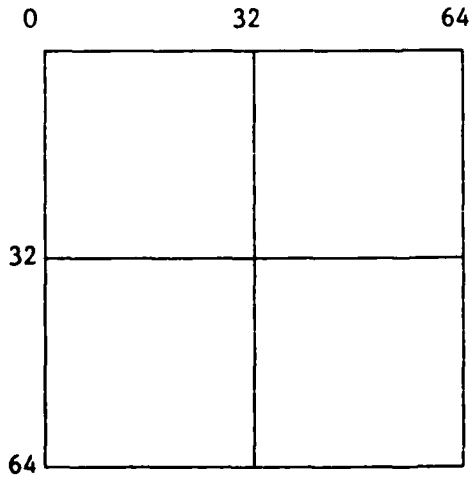
The program was tested using an example 5*5 matrix as input, obtained from QMC[1]. When the program was running correctly a random number generator was added to the DAP program. A loop was added around the assignment procedure so that timings could be obtained on the DAP Simulator for varying sizes of matrices. Copies of the program were taken and it was amended so that it ran on 24,16 and 8 bit integers and further timings taken for these.

4.2 Conversion of Assignment program for larger matrices

The previous DAP Fortran and host Fortran programs were rewritten to use matrices of data between 33*33 and 64*64. It has already been noted that the DAP consists of a 32*32 matrix of PE's. For problems which use matrices larger than the 32*32 physical array of the DAP the matrix has to be subdivided into 32*32 patches for storage, and the row and column operations of the algorithm re-organised to take account of this. This storage mode is known as Sheet or Window Mapping.

4.2.1 Sheet Mapping

Extra 32*32 matrices are set aside to store the data.
e.g.
A 64*64 matrix is held in four 32*32 matrices.



These four matrices would be defined in DAP Fortran as:

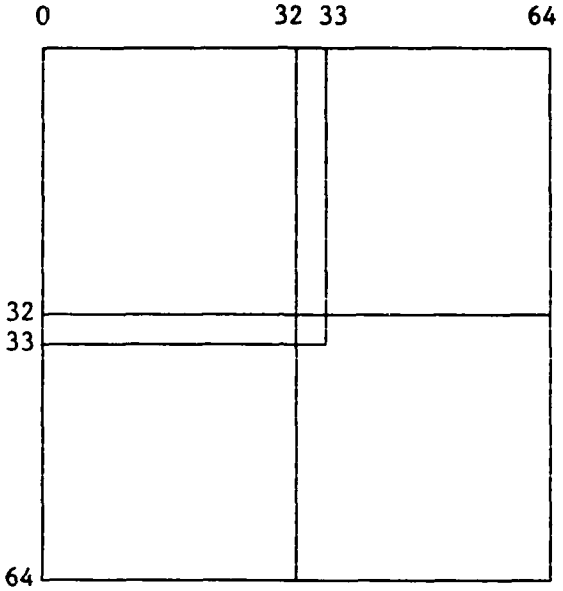
```
INTEGER*4 MAT(,,2,2)
```

and in the host program in Fortran:

```
INTEGER*4 MAT(32,32,2,2)
```

Any operation on the 64*64 matrix is applied to each of the four 32*32 matrices in turn. This introduces overheads into the execution of a DAP program.

1. Each 32*32 matrix operates sequentially.
2. If the matrix is between 33*33 and 64*64 there are unused parts of the matrices.
   e.g.
   33*33 matrix



The 33*33 matrix would execute an instruction in the following sequence.

1. Apply instruction in parallel to 1024 elements.
2. " " " " " 32 "
3. " " " " " 32 "
4. " " to 1 element.

4.2.2 Checking the Correctness of the DAP Assignment program

As before the DAP program was initially tested with the 5*5 matrix, since the steps of the program when solving this matrix had been previously worked out. The program was then tested for the case when the matrix of data overlapped the 32*32 matrices.

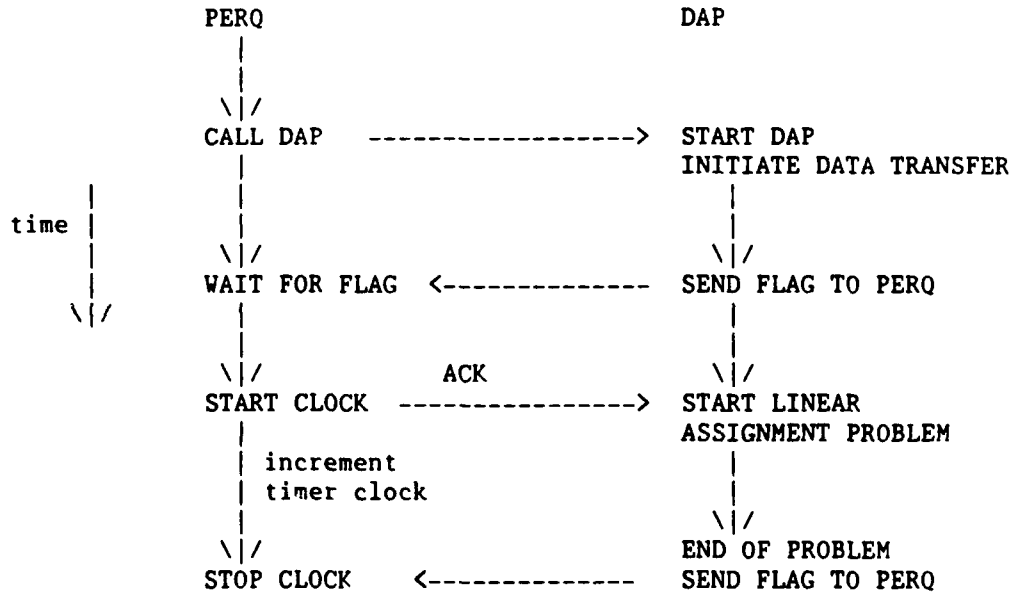
As an additional check on the results, the linear assignment problem was run on two locally available serial computers. Firstly a Pascal program running on a VAX 8600, and secondly a Fortran 77 program running on a PERQ. Both these programs solved the linear assignment problem with 32 bit integer matrices using the Bradford Method[2],[3]. The DAP program was modified to output the data matrix to a file. This file could be input to both the Pascal and Fortran assignment programs. The parallel DAP algorithm produced results identical to those from the serial Fortran 77 program, although the Pascal program showed a small 'bug' which was programmed around to give a similar check. When the program was running correctly it was amended so that it would work on matrices

up to 256*256.

4.2.3 Obtaining DAP Timings

The objective now was to obtain timings of the assignment procedure on the DAP for the larger matrices, and to compare these timings with those of a sequential computer (use of the DAP simulator would have been prohibitively slow). The timing facilities on the PERQ were used as there were procedures in C, Fortran and DAP Fortran available. The operation of these procedures is similar to the use of semaphores (see Figure 3).

FIGURE 3.
OBTAINING TIMINGS USING THE PERQ TIMER



It should be noted that the timer on the PERQ is only accurate to 17ms. To reduce the effect of any errors the times of the assignment procedure were averaged over at least 10 runs. The average timing for each size of matrix was obtained from at least 10 readings. Each time was obtained from an execution of the linear assignment problem on a particular matrix of data.

Timings were obtained on matrices whose sizes ranged from 33*33 to 256*256. The DAP program was amended to run on 16 bit integers. After being advised that the run-time checks could be switched off new timings were obtained which were about 30% faster! New timings were also obtained for 32*32 matrices without run-time checks. The gain in speed of the assignments was less significant, about 3%, because less run-time checks were originally used. Matrices up to 32*32 did not use sheet mapping, and therefore didn't require as many DO loops. The major run-time check for matrices larger than 32*32 was

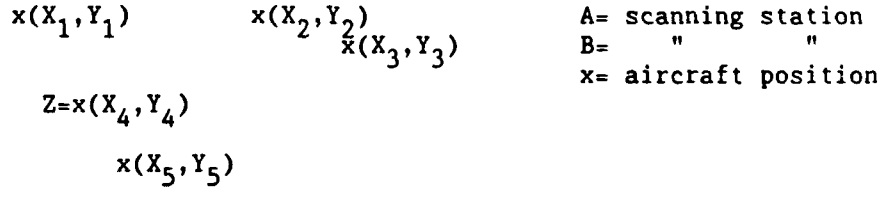
to check the control parameters of DO loops.

Timings were obtained to show the difference when the range of values in the input matrices was altered (see Table 2). The results were quite startling and illustrated the dramatic effect the data dependency of the input matrix could have upon the timings.
e.g.
A 256*256 matrix whose values were between 0 and 50 will probably find the minimum assignment straight away, since it is likely to have more than one equally low value in every row and column. If it doesn't find the 'best fit' immediately the number of iterative loops required is limited by the maximum value in the matrix, since the program contains code to repeatedly subtract values from the matrix to find the minimum assignment. If the range of values in the matrix was very large (i.e. between 0 and 100000) the number of iterative loops to find the 'best fit' is likely to be much higher.

This wide variation had not been allowed for. In 'real life' it is unlikely that the input data would be completely random so it was necessary to generate more realistic data.

The host program was amended to simulate N aircraft in a co-ordinate system with co-ordinates from 0,0 to 100,100. In this system there were two radar ststems, A and B, as shown in Figure 4.

FIGURE 4.



Each aircraft has its own set of co-ordinates ($X_1Y_1 \ldots\ldots X_NY_N$). 'A' will have a different pair of co-ordinates for aircraft Z than B because a random error margin of up to 5% either way was introduced. The differences in the co-ordinates obtained by A and B were saved in a matrix. This was the matrix that the remaining timings were obtained from. For each run a new matrix was created. Timings were taken for 32 and 16 bit signed integer matrices. It was not possible to use 8 bit integers since the created matrix had values greater than 127, thus causing numeric overflow.

4.3 Obtaining timings on a sequential computer

The host Fortran program (which set up the matrix of data and displayed the results of the assignment) was run on the VAX-8600. This program was amended

so that the matrix was saved in a file which could be read in by the Pascal program referred to in section 4.2.2. The reference program, written in Pascal and run on the VAX-8600 is one currently in use for this type of radar application. Its algorithm (based upon the Bradford method [3]) may not be strictly optimum but has been chosen with a sequential processor in mind. Both DAP and Pascal programs used integer arithmetic.

To obtain the CPU time taken by the VAX, not real time since the VAX is time shared, the control-t keys were used. This gave some information on the current process, including the CPU time used. To check the accuracy of this method the control-t keys were depressed after a delay of a few seconds with no program running. The CPU time displayed only changed by one hundredth of a second, thus providing confirmation of timing accuracy.

A second copy of the Pascal program was made and all calls to the assignment procedure removed. There were therefore two programs, one which loaded the matrix and ran the assignment procedure, and the other which only loaded the matrix. The difference between the execution times of these two programs is the time taken for the assignment.

Timings were obtained when the size of the matrix varied from 32*32 to 256*256, with all run-time checks removed (as for the DAP). Fewer timings were taken because runs were longer and the matrices of data were chosen that gave fairly average timing results on the DAP.

5. Results

5.1 Results from DAP for matrices up to 32*32

The results of the timings are included in the Appendix (see Table 1). It can be seen from the table that there is a difference in speed between 32,24,16 and 8 bit integers.

e.g.

16 bit integers took between 61% to 81% of the time taken by 32 bit integers
8 " " " " 64% " 79% " " " " " 16 " "

The difference was not greater because a significant proportion of the assignment procedure deals with logical matrices and vectors, and the speed of logical operations is unaffected by the word length of the integers.

5.2 Results from DAP for matrices between 33*33 and 256*256

The results of these timings are in the Appendix (see Table 3), in both tabular and graphic form. The relative difference between 32 and 16 bit integer arithmetic has been calculated and is included in the table. Although there are irregularites in these values it appears that as the matrix size increases, the relative difference in speed between 32 and 16 bit integers falls. As before, this difference is less than may have been expected.

e.g.

A 16 bit 33*33 matrix takes approximately 67% of the time taken for 32 bits

A 16 bit 256*256 matrix takes approximately 86% of the time taken for 32 bits

This can be explained by:

1. The larger the matrix the greater the number of DO loops, which take the same time to execute (ignoring the instructions inside the loop)
2. The larger the matrix the more searching is required to find the optimum value. This search makes heavy use of logical matrices and vectors whose execution time is unaffected by the word length of the integers used.

At each multiple of 32 in the problem size, additional matrices are required to hold the data, which gives rise to the characteristic 'step' shape as shown in the timing graphs.

5.3 Results of timings on the VAX-8600

Results of the timings are given in the Appendix (see Table 3) in graphic and tabular form. Although the timing method may appear inaccurate it should be adequate given the high values of the times involved. However there may be some small but noticeable errors for 32*32 and 64*64 matrices which had a fast execution speed.

6. Conclusion

The timings of the VAX and the DAP show that the DAP is far faster at solving the linear assignment problem for a wide range of matrix sizes. However, the speed advantage is not a smooth function of the problem size. For a 32*32 matrix the DAP is approximately 40 times faster than the VAX, yet for a 64*64 matrix it is only about 11 times faster. This is due to the problem of data being partitioned into blocks which fit the machine. Efficiency is best when the problem requires an integral number of blocks (in this case for matrices of size 32, 64, 96 etc), but the speed advantage over the VAX-8600 is typically an order of magnitude, rising to two orders for matrices of 256*256.

Despite the inefficiencies of partitioning the problem the DAP still greatly outperforms the VAX. As the matrices become larger the difference in times between the two computers becomes even more marked. This is most noticeable when the matrix becomes larger than 160*160.
e.g.

For a 64*64 matrix the DAP is approximately 11 times faster than the VAX
" " 256*256 " " " " " 114 " " " " "

The DAP is made up of 1024 relatively simple and slow processing elements (PE's). Its large speed advantage is due to the parallelism that is used in solving the assignment problem. The linear assignment problem involves many simple arithmetical, logical or bit operations that can be applied concurrently in each PE.

7. Appendix

7.1 Average Times for Matrices from 2*2 to 32*32 on the DAP

Table 1

These timings were from a program which:

1. Had run-time checks.
2. Used randomly generated data, max value 100.

The average timings for 32 bits were obtained from:

- 10 measurements for matrices of size 2*2 to 10*10
- 20 " " " " " 11*11 " 32*32

The average timings for 24 bits were from:

- 10 measurements for matrices of size 2*2 to 14*14
- 20 " " " " " 15*15 " 32*32

The average timings for 16 and 8 bits were obtained from 20 runs for all matrix sizes.
Times are given in milliseconds.

MATRIX SIZE	32 bits	24 bits	16 bits	8 bits
2	1.636	1.306	1.003	0.696
3	1.636	1.366	1.107	0.795
4	1.823	1.551	1.267	0.879
5	2.034	1.678	1.262	0.906
6	3.376	1.874	1.481	1.108
7	2.403	2.060	1.581	1.195
8	2.440	2.308	1.613	1.276
9	2.679	2.278	1.804	1.359
10	2.905	2.451	1.925	1.554
11	2.956	2.674	1.925	1.554
12	3.255	2.886	2.296	1.729
13	3.094	2.770	2.339	1.853
14	3.453	2.918	2.404	1.896
15	3.785	3.139	2.610	1.974
16	4.016	3.309	2.795	2.033
17	4.345	3.528	3.009	2.228
18	4.647	3.660	3.101	2.258
19	4.386	3.694	3.186	2.250
20	4.545	3.804	3.368	2.312
21	4.514	3.843	3.333	2.279
22	4.513	3.782	3.387	2.393
23	4.640	3.885	3.414	2.447
24	4.960	4.245	3.792	2.423
25	5.009	4.306	3.798	2.620
26	5.073	4.287	3.826	2.614
27	5.093	4.482	3.942	2.714
28	5.203	4.481	4.032	2.710
29	5.311	4.600	4.021	2.802
30	5.315	4.703	3.993	2.852
31	5.133	4.538	4.067	2.861
32	5.218	4.698	4.265	2.948

7.2 Timings for 256*256 32 bit Matrix with varying range of Input Data

Table 2

Times obtained with randomly generated data and no run-time checks.

MAX NUMBER	TIME (SECONDS)
50	0.665
150	1.327
450	1.760
950	2.195
2000	3.277
5000	4.617
10000	6.975
50000	9.062
100000	10.477
200000	10.637

7.3 Average Timings for Matrices between 32*32 and 256*256 on the DAP and VAX

Table 3

SIZE OF MATRIX	DAP (1) 32 BITS	DAP (1) 16 BITS	RELATIVE DIFFERENCE	AVERAGE(2) VAX TIME 32 BITS	SPEED UP OF DAP OVER VAX (NUMBER OF TIMES FASTER)
32	0.005			0.20	40
33	0.058	0.039	0.67	0.19	3
48	0.074	0.052	0.70		
64	0.092	0.068	0.74	1.05	11
65	0.187	0.141	0.75	1.10	5
80	0.193	0.148	0.77		
96	0.216	0.168	0.78	2.42	11
97	0.344	0.278	0.81	2.24	6
112	0.348	0.269	0.77		
128	0.349	0.270	0.77	4.96	14
129	0.599	0.453	0.76	4.70	7
144	0.651	0.531	0.81		
160	0.685	0.574	0.84	9.34	13
161	0.952	0.781	0.82	9.23	9
176	1.063	0.864	0.81		
192	1.081	0.897	0.83	56.92	52
193	1.389	1.167	0.84	61.35	44
208	1.450	1.253	0.86		
224	1.452	1.203	0.83	125.62	86
225	1.858	1.464	0.79	128.86	69
240	1.905	1.664	0.87		
256	1.772	1.645	0.86	202.96	114

(1). The DAP timings:

1.1 Do not use run-time checks.
1.2 Use radar simulated data.
1.3 Are the average of

10	measurements	for	32	bit	matrices	of	size	33*33	to 192*192
15	"	"	"	"	"	"	"	193*193	" 256*256
10	"	"	16	"	"	"	"	33*33	" 256*256

(2). The VAX times:

2.1 Are the average of 4 measurements.
2.2 Do not use run-time checks.
2.3 Use the same radar simulated data as the DAP.

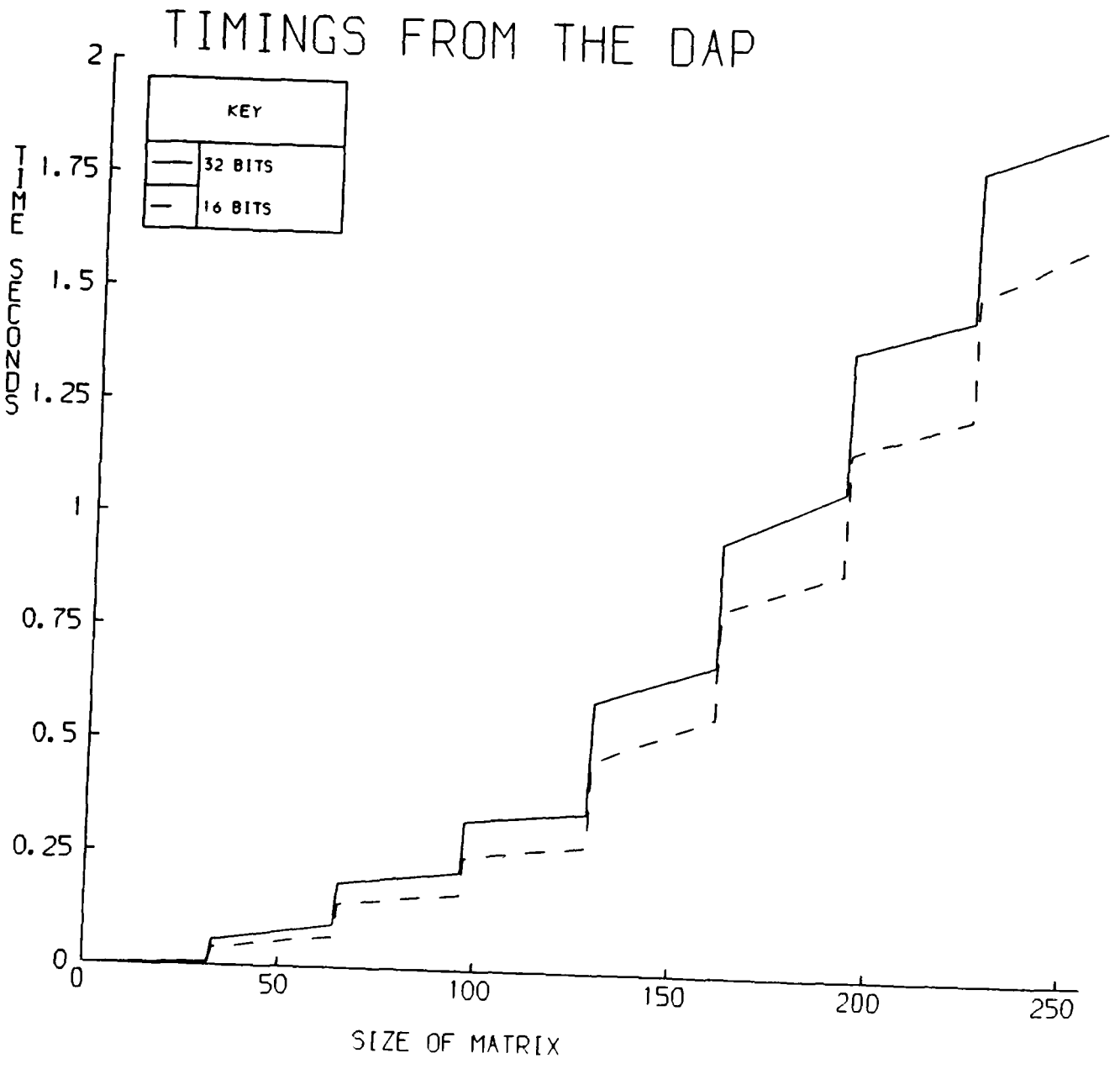
8. References

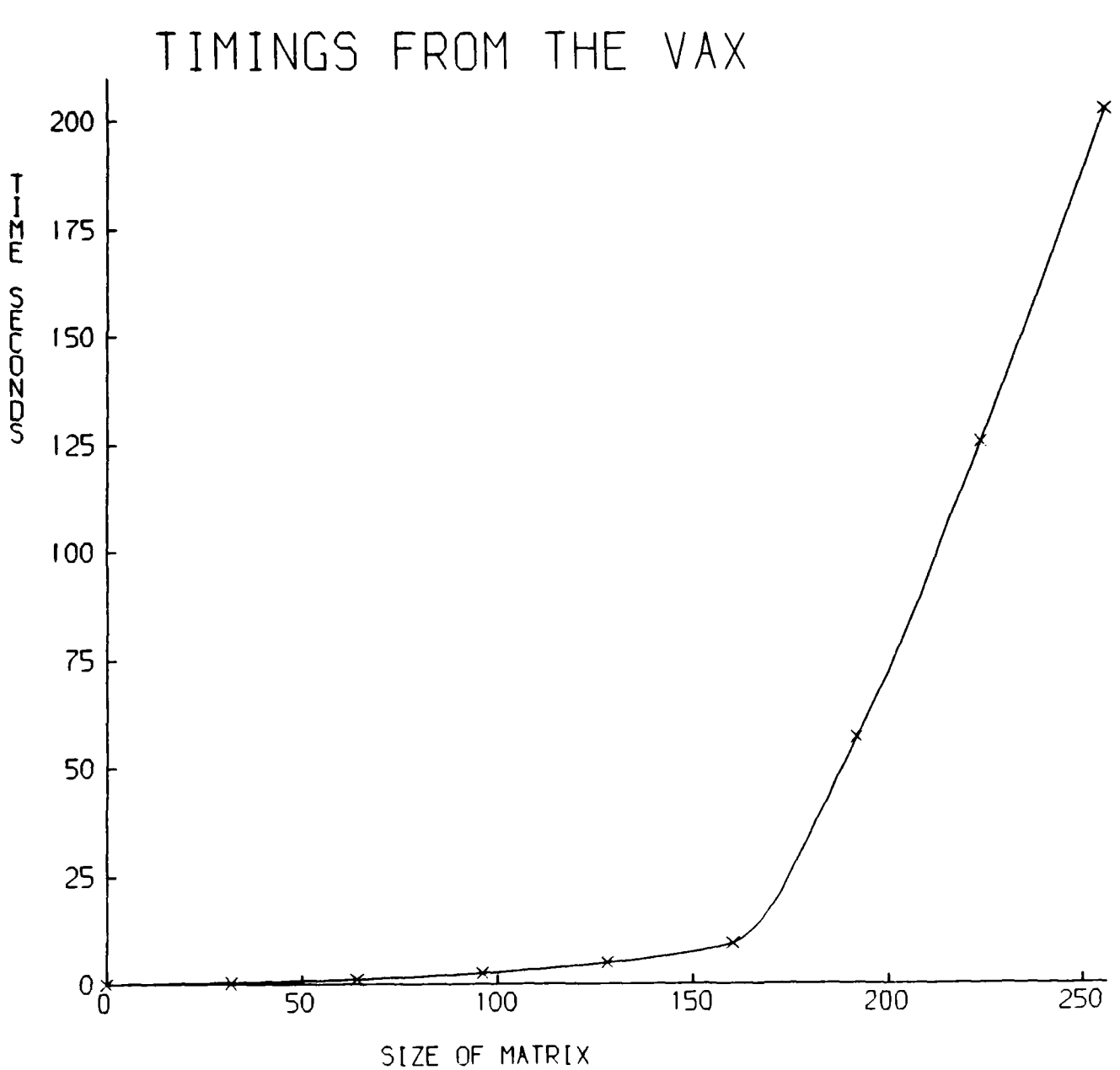
[1] J.Yadegar,A.Frieze and S.T.Davies, H01_L_ASSIGN. Subroutine to solve the linear assignment problem. DAP Subroutine library. Queen Mary College, London.

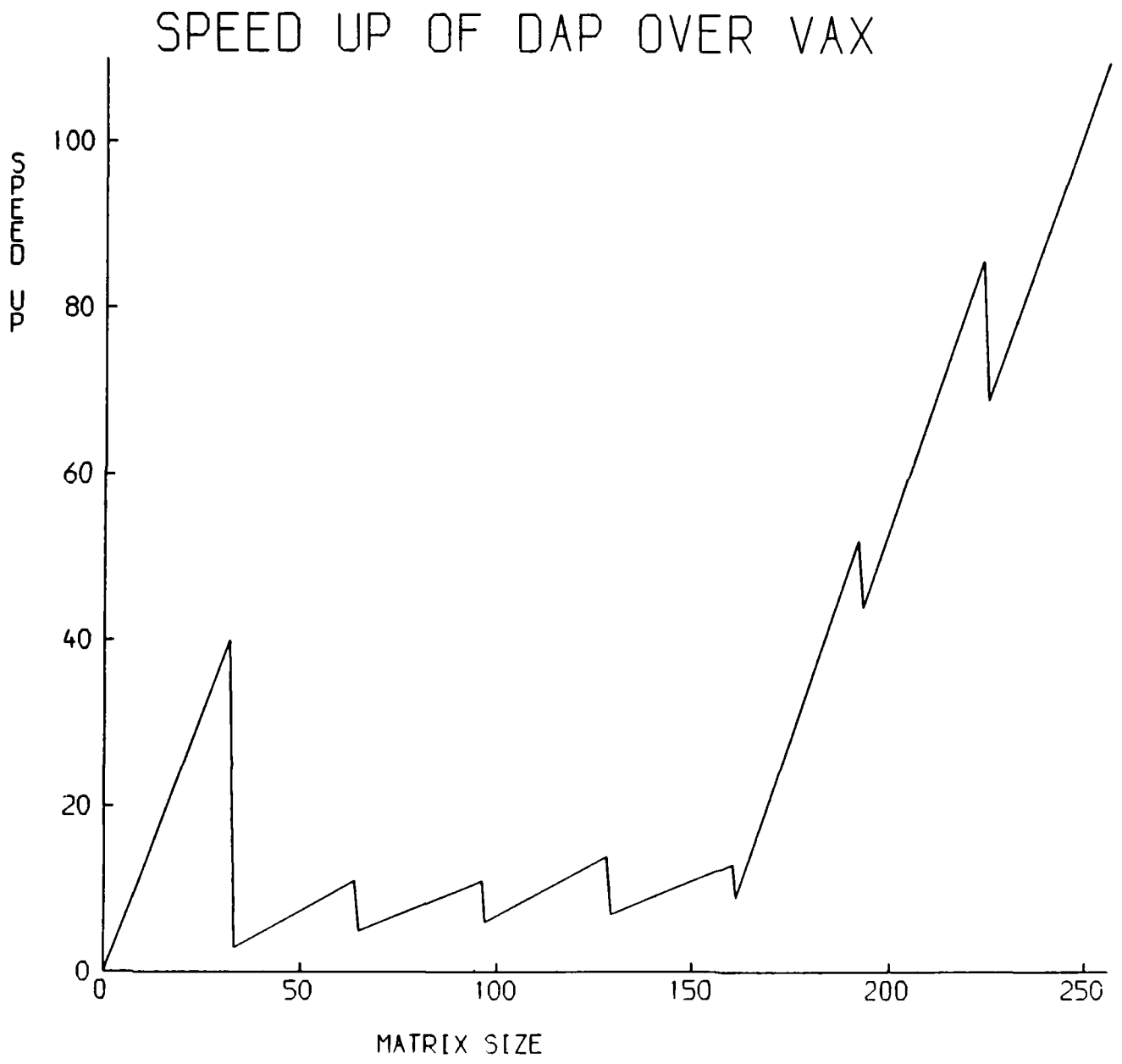
[2] S.Magowan. A method of plot to track correlation
RRE Memorandum 2153, January 1965

[3] C.Mack. Bradford method, Bradford Institute of Technology.

THIS PAGE IS LEFT BLANK INTENTIONALLY


TIMINGS FROM THE DAP
KEY
32 BITS
16 BITS
TIME SECONDS
2
1.75
1.5
1.25
1
0.75
0.5
0.25
0
0
50
100
150
200
250
SIZE OF MATRIX


TIMINGS FROM THE VAX
TIME SECONDS
200
175
150
125
100
75
50
25
0
0
50
100
150
200
250
SIZE OF MATRIX


SPEED UP OF DAP OVER VAX
SPEED UP
100
80
60
40
20
0
0
50
100
150
200
250
MATRIX SIZE

THIS PAGE IS LEFT BLANK INTENTIONALLY

DOCUMENT CONTROL SHEET

Overall security classification of sheet ..UNCLASSIFIED.....

(As far as possible this sheet should contain only unclassified information. If it is necessary to enter classified information, the box concerned must be marked to indicate the classification eg (R) (C) or (S))

1. DRIC Reference (if known)	2. Originator's Reference Memorandum 4213	3. Agency Reference	4. Report Security Classification Unclassified
5. Originator's Code (if known) 778400	6. Originator (Corporate Author) Name and Location Royal Signals and Radar Establishment St Andrews Road, Malvern, Worcestershire WR14 3PS		
5a. Sponsoring Agency's Code (if known)	6a. Sponsoring Agency (Contract Authority) Name and Location		

7. Title
THE APPLICATION OF A DISTRIBUTED ARRAY PROCESSOR (DAP) TO LINEAR ASSIGNMENT PROBLEMS IN RADAR TRACKING

7a. Title in Foreign Language (in the case of translations)

7b. Presented at (for conference papers) Title, place and date of conference

8. Author 1 Surname, initials	9(a) Author 2	9(b) Authors 3,4...	10. Date	pp. ref.
Stanley A J	Simpson P	Roberts R B G	7.88	18

11. Contract Number	12. Period	13. Project	14. Other Reference

15. Distribution statement
Unlimited

Descriptors (or keywords)

continue on separate piece of paper

Abstract
A Distributed Array Processor (DAP) is an SIMD parallel processing machine composed of 1024 one-bit processing elements (PEs). This Memorandum examines the application and detailed performance of this machine to the linear assignment problem with data arrays up to 256x256 in size. The linear assignment problem is used in ESM, radar tracking, and other fields where it is necessary to assign data from two or more classes to each other. Since the assignment problem is solved by a computationally intensive algorithm a comparison is made between the DAP and a serial machine, a VAX 8600, to assess the speed gains obtained from the DAP by executing instructions in parallel. The results show that the DAP is far faster at solving this problem than the VAX by up to two orders of magnitude.