

Purdue University

Purdue e-Pubs

Department of Computer Science Technical
Reports

Department of Computer Science

1993

Some Experiments with A Basic Linear Algebra Routine on Distributed Memory Parallel Systems

H. Byun

Elias N. Houstis

Purdue University, enh@cs.purdue.edu

E. A. Vavalis

Report Number:

93-034

Byun, H.; Houstis, Elias N.; and Vavalis, E. A., "Some Experiments with A Basic Linear Algebra Routine on Distributed Memory Parallel Systems" (1993). *Department of Computer Science Technical Reports*. Paper 1051.

<https://docs.lib.purdue.edu/cstech/1051>

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries.
Please contact epubs@purdue.edu for additional information.

**Some Experiments with a
Basic Linear Algebra Routine
on Distributed Memory
Parallel Systems**

H. Byun, E.N. Houstis
and E.A. Vavalis

CSD-TR-93-034
June 1993

SOME EXPERIMENTS WITH A BASIC LINEAR ALGEBRA ROUTINE ON DISTRIBUTED MEMORY PARALLEL SYSTEMS *

H. BYUN, E.N. HOUSTIS AND E.A. VAVALIS †

Abstract

In this paper we describe the algorithm used, discuss the implementation and present the performance of the Basic Linear Algebra Subroutine (BLAS) `sgemv` for distributed-memory multiprocessors. The basic assumption is that the matrix and the vectors are row distributed among processors. Performance data from nCUBE II, iPSC/860 and iPSC DELTA machines are presented.

1. Introduction. In this study we present data that describe various aspects of the performance of the Basic Linear Algebra Subroutine (BLAS) `sdgemv` on three distributed memory multiprocessor systems namely the nCUBE II, the iPSC/860 and the iPSC DELTA. `sdgemv` is a member of a set of parallel BLAS routines we have implemented on such machines [1]. The software methodology utilized to parallelize the BLAS routines assumes that each processor performs the appropriate local operations by calling the corresponding uniprocessor BLAS routines ([9], [5], [4]). The local results are "combined" by PICL [6] routines to generate the final answer. It is worth noticing that the global combine operations can use any multiprocessor connection topology that PICL supports.

The rest of the paper is organized as follows. Section 3 consists of a list of tables that present raw timing data measuring the total elapse and total (communication and idle) overhead time required by the `sdgemv` routine to perform the matrix-vector operations on the three machines considered. Using the data given in Section 3 we present, in Section 4 the Gflops achieved on the three machines for different matrix sizes and connection topologies, and in Section 5 the data that show the differences observed when we used the optimized uniprocessor BLAS routines instead of FORTRAN BLAS on the iPSC/860. Finally in Section 6 we give the utilization and concurrency profiles and in Section 7 spacetime execution diagrams. The data in the latter two sections were obtained using PARAGRAPH [7].

2. The algorithm and its implementation. In this section we discuss the parallel implementation of the matrix \times vector operations Ax or $A^T x$. Throughout, we assume that the matrix A and the vector x is distributed among processors row-wise. This distribution is defined by a vector $idist(i), i = 1, nprocs + 1$ where $nprocs$ is the number of processors and $idist(i)$ denotes the global index of the first row of A

* This work was partially supported by NSF grants 9123502-CDQ and 9202536-CCR, AFOSR F49620-92-J-0069 and PRF 6902003. This research was performed in part using the Intel Touchstone Delta System operated by Caltech on behalf of the Concurrent Supercomputer Consortium. Access to this facility was provided by Purdue University.

† Purdue University, Computer Science Department, West Lafayette, IN 47907.

(first element of x) that belongs to processor i . Thus, we store rows from $idist(i)$ to $idist(i+1) - 1$ of the matrix A on the local memory of processor i together with the associated elements of x . $idist$ is the only global information needed and all other variables are local to each processor.

For the implementation of the Ax operation, we follow the methodology described in the previous section. As an example we give in Figure 1 the *actual* code for the `sdgemv` routine for full matrices. We assume that the data are distributed on a wrap around linear array of processors. The level two BLAS routine `sgemv` gets as input the matrix $A \in R^{m \times n}$, the vectors $x, y \in R^m$ and the scalars α and β . It computes the vector $y = \alpha Ax + \beta y$ or $\alpha A^T x + \beta y$. In the case of matrix A , each node calculates the part of the product that involves local data by utilizing the uniprocessor level two BLAS routine `sgemv`. Then it broadcasts its local vector x to all other processors by calling the PICL routine `bcast1`, in which the reception and forwarding of the message are decoupled. Thus, processors that participate in this broadcast call `sgemv` which computes the part of the product associated with the incoming vector x before forwarding it. Finally, we restore the original value of the vector x by reading it from a nearest neighbor. It is important to notice that the only memory overhead for the routine `sdgemv` is the integer array $idist$ of length $nprocs + 1$.

In the case of the $A^T x$ operation, each processor i calls `sgemv` to compute $\alpha A^T_i x$ and stores this result to a buffer. The entries of all these buffers are added component-wise using the PICL routine `gsum0` and the result is stored in the buffer at processor 1. Finally a call to the level one BLAS routine `saxpy` is used to accumulate the term βy to the buffer. The above described procedure is repeated $nprocs - 1$ more times as shown in Figure 1.

We have experiment with two different interconnection topologies that PICL supports, namely the ring connectivity and the full connectivity. It is worth noticing that in the case of banded or sparse matrices we were able to successfully follow the above approach, coupled with appropriate data structures, with only a few basic differences [2].

3. Raw time data. The uniprocessor BLAS routines available to us on the iPSCs [8] were used unless it is stated otherwise in the caption of the tables. All times are in milli-seconds.

4. Gflops achieved. In Figures 2 and 3 we present the Gflops achieved on the three machines using ring topology for both the non-transposed and transposed cases respectively. It is worth noticing that the "theoretical peak" (determined by counting the number of floating point operations in full precision that can be completed during a cycle) for the nCUBE II, the iPSC 860 and for the iPSC DELTA is .15, 2.6 and 20 Gflops respectively [3].

5. The affect of using optimized BLAS. In Figure 4 we present the speedup achieved on the iPSC/860 by using optimized uniprocessor BLAS routines [8] for the non-transposed case. As we see the performance increases four to six times. It is also apparent that the performance drops as the number of processors increases. To further

FIG. 1. The fortran code of subroutine sagemv.

```

subroutine sagemv(trans,m,n,alpha,a,lda,x,beta,y,tmp,idst)

integer idst(1)
real    a(lda,1),x(1),y(1),tmp(1)
character trans

integer nprocs, me, host, top, ord, dir
common /open/ nprocs,me,host
common /setarc/ top, ord, dir

integer bytes, mtype, root, lnx, jidx, node_no

lnx = idst(me+2) - idst(me+1)
bytes = 4 * lnx

if( ( lsame(trans,'N') ) .or. ( lsame(trans,'n') ) ) then
  jidx = idst(me+1)
  call sgemv(trans,m,n,alpha,a(1,jidx),lda,x,1,beta,y,1)
  mtype = 1000 + me
  call bcast0(x,bytes,mtype,me)
  do 100 i=1,nprocs-1
    root = mod(me-i+nprocs, nprocs)
    mtype = 1000 + root
    jidx = idst(root+1)
    call bcast0(x,bytes,mtype,root)
    call sgemv(trans,m,n,alpha,a(1,jidx),lda,x,1,1.,y,1)
100  continue
    root = mod(me+1+nprocs, nprocs)
    call send0(x,bytes,root+1000,root)
    call recv0(x,bytes,me+1000)
  endif
if( ( lsame(trans,'T') ) .or. ( lsame(trans,'t') ) ) then
  mtype = 4000
  do 200 node_no = 0, nprocs-1
    istart = idst(node_no+1)
    call sgemv(trans,m,n,alpha,a(1,istart),lda,x,1,0.,tmp,1)
    call gsum0(tmp,lnx,4,mtype,node_no)
    if (node_no .eq. me) then
      if (beta .ne. 0.) call saxpy(lnx,beta,y,1,tmp,1)
    call scopy(lnx,tmp,1,y,1)
  endif
200  continue
endif

return
end

```

TABLE 1

The total elapse and overhead timings on the nCUBE II for matrices A of size $n \times n$.

n	1			2			4			8			16			32			64			
	T_e	T_c	T_o	T_e	T_c	T_o	T_e	T_c	T_o	T_e	T_c	T_o	T_e	T_c	T_o	T_e	T_c	T_o	T_e	T_c	T_o	
800	2834	1424	3	717	2	366	3															
1200		3194	3	1605	3	812	3	418	6													
1600				2846	3	1435	4	732	6	386	12											
2400						3210	5	1624	7	836	12	454	22									
3200								2869	8	1463	12	771	24									
4800										3248	14	1672	23									
6400												2926	25									

TABLE 2

The total elapse and overhead time on the nCUBE II for matrices A^T of size $n \times n$.

n	1			2			4			8			16			32			64			
	T_e	T_c	T_o	T_e	T_c	T_o	T_e	T_c	T_o	T_e	T_c	T_o	T_e	T_c	T_o	T_e	T_c	T_o	T_e	T_c	T_o	
800	2615	1316	3	666	3	343	5															
1200		2950	4	1487	4	756	6	395	6													
1600				2632	6	1333	7	686	10	370	15											
2400						2974	9	1514	12	790	17	442	27									
3200								2666	14	1373	19	749	30									
4800										3028	24	1581	36									
6400												2747	40									

TABLE 3

The total elapse and overhead timings on the iPSC/860 for matrices A of size $n \times n$. This implementation is based on full connectivity.

n	1			2			4			8			16			32			64			
	T_e	T_c	T_o	T_e	T_c	T_o	T_e	T_c	T_o	T_e	T_c	T_o	T_e	T_c	T_o	T_e	T_c	T_o	T_e	T_c	T_o	
1600	201	106	4	60	4	38	6															
2400		234	6	125	5	74	11	53	11													
3200				214	8	122	12	80	16	63	24											
4800						254	13	151	22	110	27	92	43									
6400								250	27	162	36	129	52									
9600										302	48	220	63									
12800												310	85									

TABLE 4

The total elapse and overhead timings on the iPSC/860 for matrices A of size $n \times n$. This implementation is based on ring topology.

n	1		2		4		8		16		32		64	
	T_e	T_e	T_o	T_e	T_o	T_e	T_o	T_e	T_o	T_e	T_o	T_e	T_o	
1600	201	106	3	59	6	38	6							
2400		233	6	124	5	72	6	52	11					
3200				213	8	118	8	76	9	62	22			
4800						249	10	145	13	105	16	92	34	
6400								238	16	152	24	125	42	
9600										281	19	207	41	
12800												286	43	

TABLE 5

The total elapse and overhead timings on the iPSC/860 for matrices A of size $n \times n$. This implementation is based on full connectivity and the uni-processors level two BLAS used were written in FORTRAN.

n	1		2		4		8		16		32		64	
	T_e	T_e	T_o	T_e	T_o	T_e	T_o	T_e	T_o	T_e	T_o	T_e	T_o	
3200				1873	6	924	10	468	10	251	22			
4800						2087	12	1042	16	537	17	303	34	
6400								1849	10	937	17	502	33	
9600										2072	21	1069	34	
12800												1852	42	

TABLE 6

The total elapse and overhead timings on the iPSC/860 for matrices A of size $n \times n$. This implementation is based on ring connectivity and the uni-processors level two BLAS used were written in FORTRAN.

n	1		2		4		8		16		32		64	
	T_e	T_e	T_o	T_e	T_o	T_e	T_o	T_e	T_o	T_e	T_o	T_e	T_o	
3200				1741	8	870	16	455	23	260	35			
4800						1957	34	994	34	535	41	330	74	
6400								1744	35	912	51	523	81	
9600										1987	60	1071	85	
12800												1825	90	

TABLE 7

The total elapse and overhead timings on the iPSC/860 for matrices A^T of size $n \times n$. This implementation is based on full connectivity.

n	1		2		4		8		16		32		64	
	T_e	T_e	T_o	T_e	T_o	T_e	T_o	T_e	T_o	T_e	T_o	T_e	T_o	
1600	198	104	3	59	4	36	7							
2400		230	6	124	8	69	8	48	9					
3200				197	7	114	8	76	18	63	20			
4800						250	13	145	17	101	22	101	39	
6400								239	20	157	16	131	38	
9600										293	35	204	47	
12800												310	49	

TABLE 8

The total elapse and overhead timings on the iPSC/860 for matrices A^T of size $n \times n$. This implementation is based on ring topology.

n	1		2		4		8		16		32		64	
	T_e	T_e	T_o	T_e	T_o	T_e	T_o	T_e	T_o	T_e	T_o	T_e	T_o	
1600	198	104	3	62	7	44	6							
2400		230	7	127	12	81	14	61	19					
3200				218	13	129	20	93	30	84	39			
4800						262	26	166	37	127	47	137	77	
6400								264	46	188	62	170	87	
9600										337	73	254	99	
12800												378	110	

TABLE 9

Total elapse and overhead measured times the DELTA for matrices A of size $n \times n$.

n	1		4		8		16		32		64		128		256		512	
	T_e	T_e	T_o	T_e	T_o	T_e	T_o	T_e	T_o	T_e	T_o	T_e	T_o	T_e	T_o	T_e	T_o	
1600	100	31	2	22	3													
2400		62	3	39	5	32	6											
3200		104	6	62	3	44	7	36	8									
4800				125	5	79	7	62	13	51	20							
6400						125	10	86	11	71	21	83	38					
9600								163	18	129	30	107	41	141	120			
12800										182	31	152	50	176	113	317	224	
19200												277	81	226	114	329	233	
25600														325	116	412	253	
38400																548	343	

TABLE 10
 Total clapse and overhead measured times on the DELTA for matrices A^T of size $n \times n$.

n	1		4		8		16		32		64		128		256		512		
	T_e	T_o	T_e	T_o	T_e	T_o	T_e	T_o	T_e	T_o	T_e	T_o	T_e	T_o	T_e	T_o	T_e	T_o	
1600	78		27	3	21	4													
2400			54	4	36	5	27	6											
3200			90	6	56	5	40	8	36	9									
4800					109	9	73	6	54	14	58	18							
6400							112	13	83	14	73	16	77	29					
9600									146	21	109	26	115	32	153	55			
12800											166	25	148	35	167	58	255	110	
19200													227	54	241	60	288	112	
25600															312	68	331	113	
38400																	489	126	

examine this we present in Figure 5 the total overhead time involved. This overhead measures communication and synchronization time and obviously should not depend on how we perform the floating point arithmetic. As the data in Figure 5 surprisingly show that this is not correct since the use of optimized BLAS might double or half the overhead time depending on the interconnection topology. This is due to cash memory management that both the BLAS and the communication router use. We will present a complete analysis of that phenomenon elsewhere.

5.1. Concurrency and utilization diagrams. In Figures 6 and 7 we present the concurrency profile and the utilization summary diagrams we obtained using PARAGRAPH on the 64 node nCUBE II and iPSC/860 respectively. The matrix sizes $n = 6400$ and $n = 12800$ correspond to the largest problem we could fit on these machines. Notice that the nCUBE II machine we use is configured with 4 Mbytes of memory per node while the iPSC/860 with 16 Mbytes per node.

It is worth noticing the erratic utilization behavior on the iPSC/860. This is due to the non-deterministic routing message mechanism. The full analysis of this behavior is under way and it will be given elsewhere.

5.2. Spacetime diagrams. In Figures 8 and 9 we give the spacetime execution diagrams associated with the problems and machine configurations considered in Figures 6 and 7 respectively. We only give the final time intervals since initially (intervals [0,121], [0-70] [0-90] and [0-70]) all processors are doing only floating point operations calculating the local part of the matrix vector operation. Notice that the erratic utilization behavior on the iPSC/860 mentioned in the previous section can be noticed here too.

REFERENCES

- [1] H. BYUN, E. HOUSTIS, AND E. VAVALIS, *ROWDLAS User's Guide*, Tech. Report CSD-TR-93-017, Purdue University, W. Lafayette, IN, Jan. 1993.
- [2] H. BYUN, S. KORTESIS, E. HOUSTIS, AND E. VAVALIS, *A virtual parallel environment for implementing neural network computations on parallel machines*, *Neural, parallel and scientific computations*, 3 (1994), p. to appear.
- [3] J. DONGARRA, *Performance of various computers using standard linear equation software*, Tech. Report CS-89-85, University of Tennessee, Knoxville, TN, Feb. 1993.
- [4] J. J. DONGARRA, J. DU CROZ, I. DUFF, AND S. HAMMARLING, *A set of level 3 basic linear algebra subprograms*, *ACM Trans. Math. Softw.*, 16 (1990), pp. 1-17.
- [5] J. J. DONGARRA, J. DU CROZ, S. HAMMARLING, AND R. J. HANSON, *An extended set of basic linear algebra subprograms: Model implementation and test programs*, *ACM Trans. Math. Softw.*, 14 (1988), pp. 18-32.
- [6] G. GEIST, M. HEATH, B. PEYTON, AND P. WORLEY, *A users' guide to PICL a portable instrumented communication library*, Tech. Report ORNL/TM-11616, Mathematical Sciences Section, Oak Ridge National Laboratory, Oak Ridge, TN, Oct. 1990.
- [7] M. HEATH AND J. ETHERIDGE, *Visualizing the performance of parallel programs*, *IEEE Software*, 8 (1991), pp. 29-39.
- [8] INTEL CORPORATION, *iPSC/860 Basic Math Library User's Guide*, 1st ed., April 1991.
- [9] C. L. LAWSON, R. J. HANSON, D. R. KINCAID, AND F. T. KROGH, *Basic linear algebraic subprogram for Fortran usage*, *ACM Trans. Math. Softw.*, 5 (1979), pp. 324-325.

FIG. 2. The achieved Gflops for $A \in R^{n \times n}$.

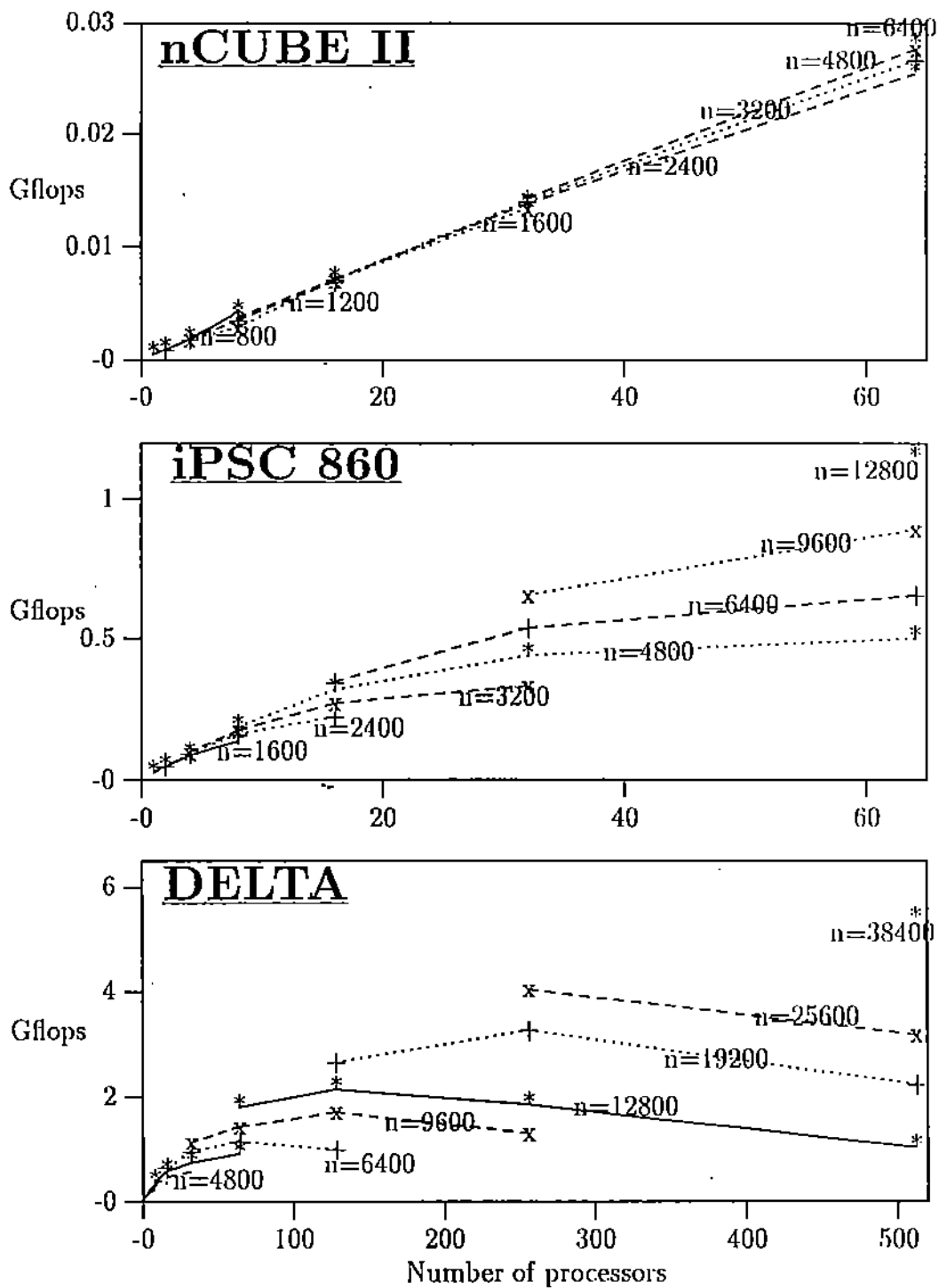


FIG. 3. The achieved Gflops for $A^T \in R^{n \times n}$.

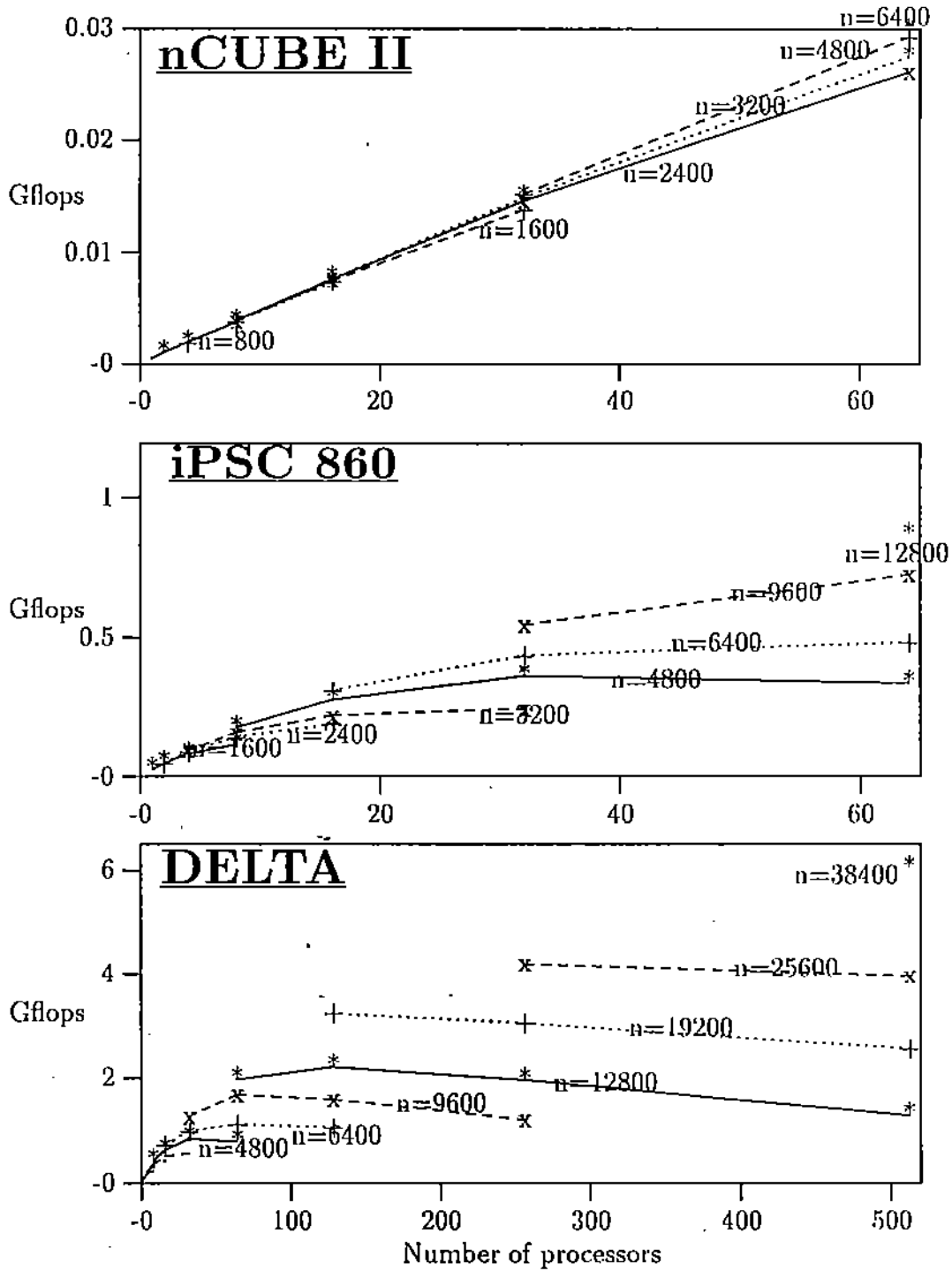


FIG. 4. The achieved speedup on the iPSC/860 with full (F) and ring (R) connectivity and $A \in R^{n \times n}$.

FIG. 5. The total overhead time (in milli-seconds) on the iPSC/860 with full (F) and ring (R) connectivity, FORTRAN and optimized uniprocessor BLAS routines and $A \in R^{n \times n}$.

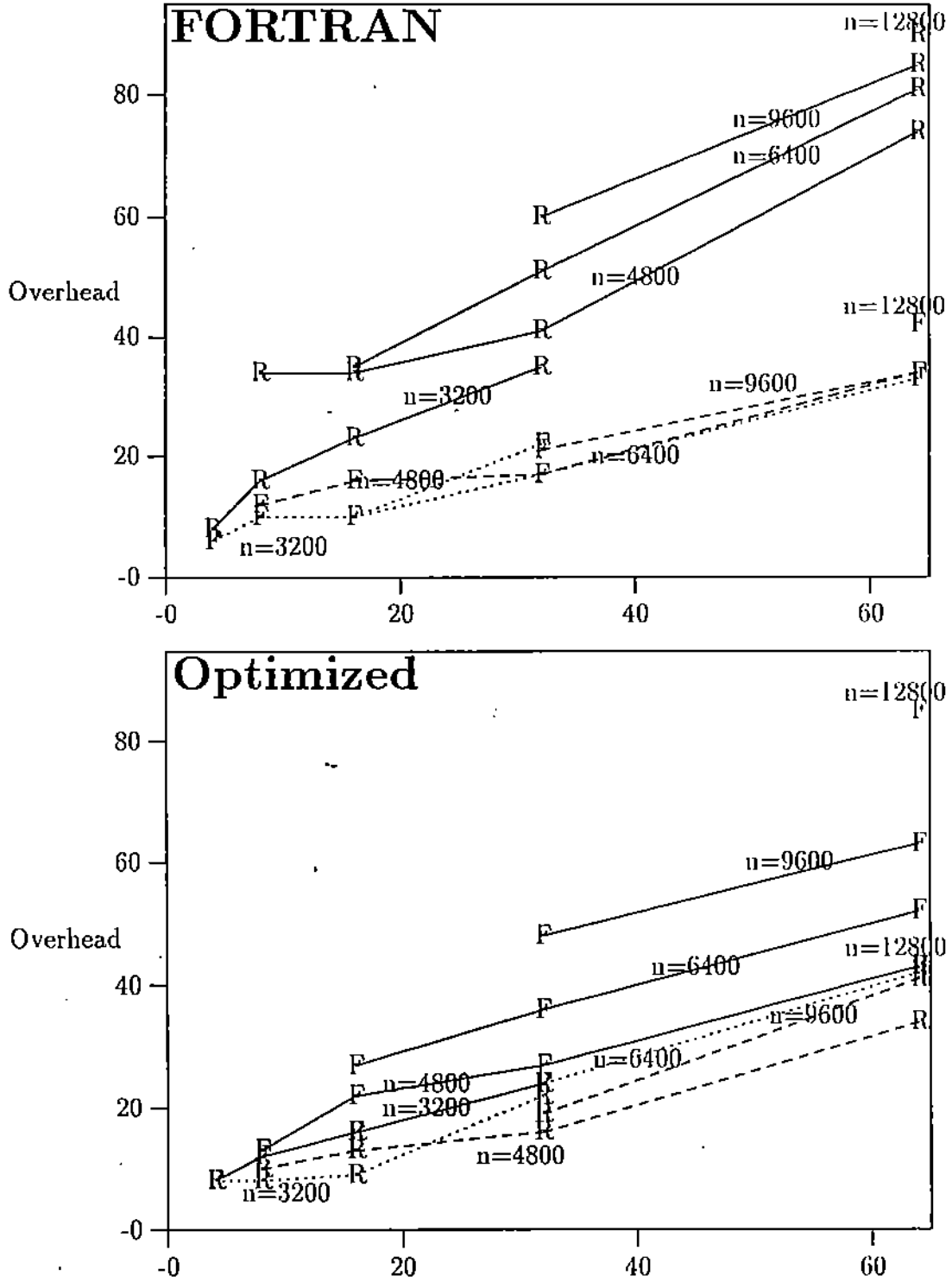


FIG. 6. Profile and utilization diagrams on the 64 node nCUBE II for matrices of size $n = 6400$. Ring connectivity were used.

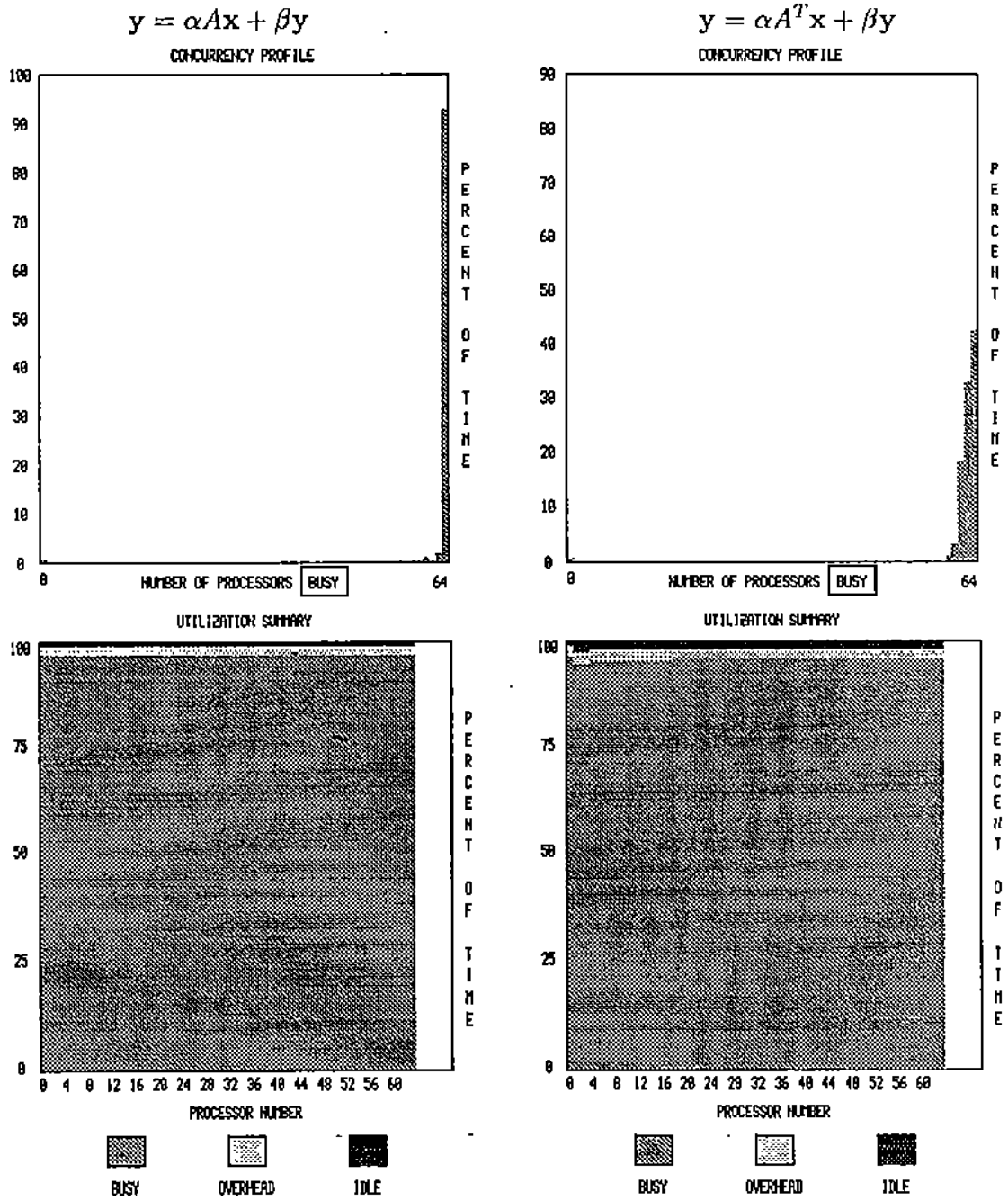


FIG. 7. Profile and utilization diagrams on the 64 node iPSC/860 for matrices of size $n = 128000$. Ring connectivity were used.

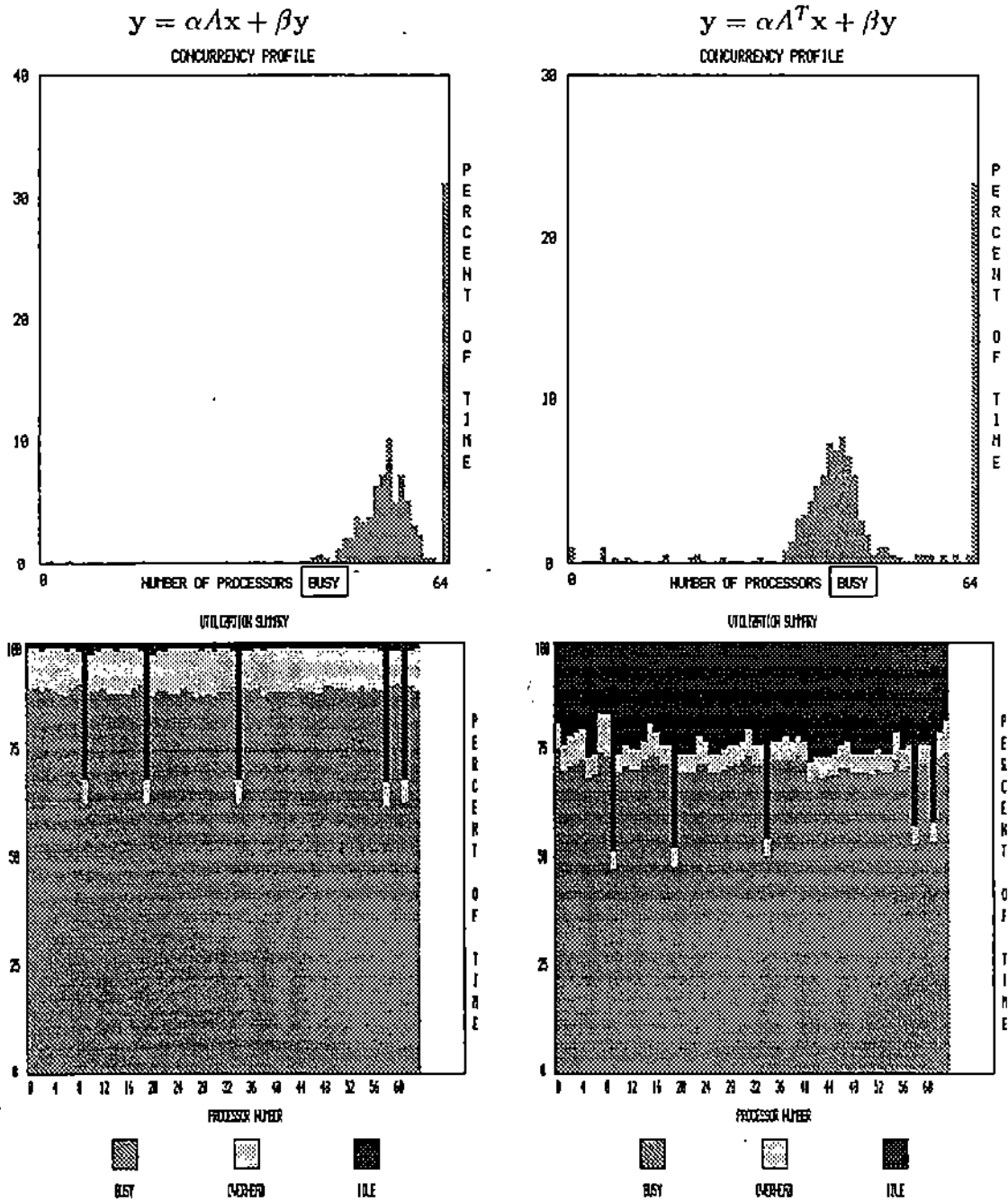
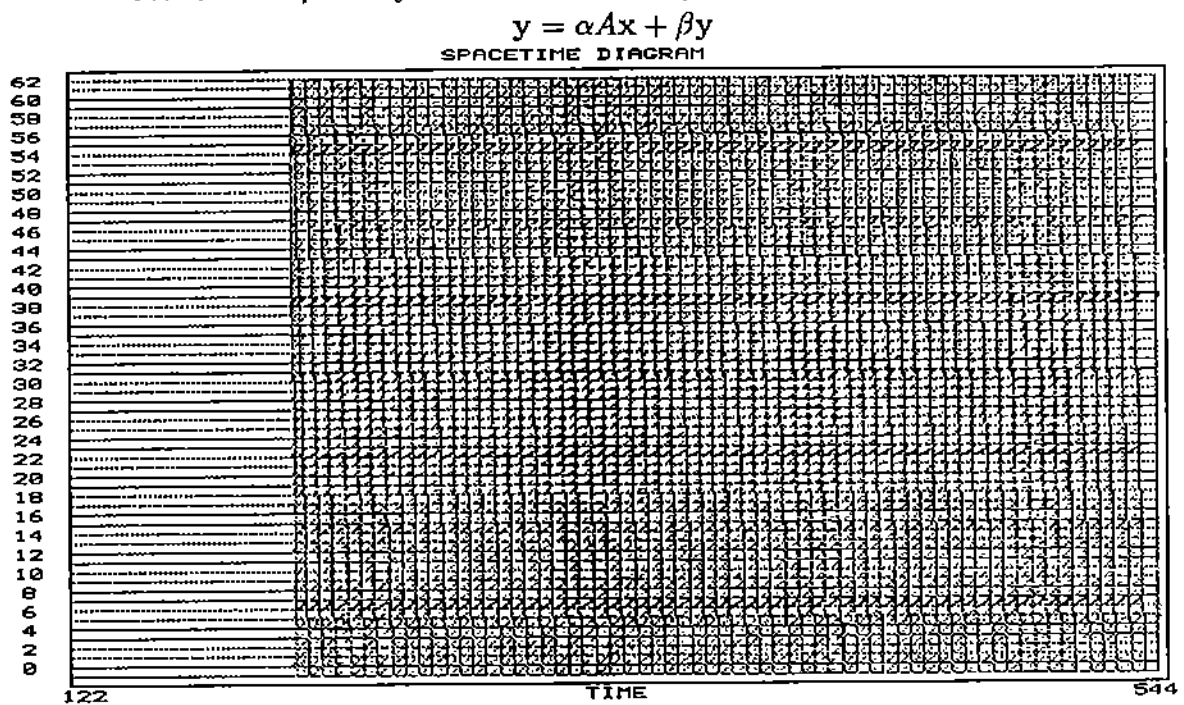
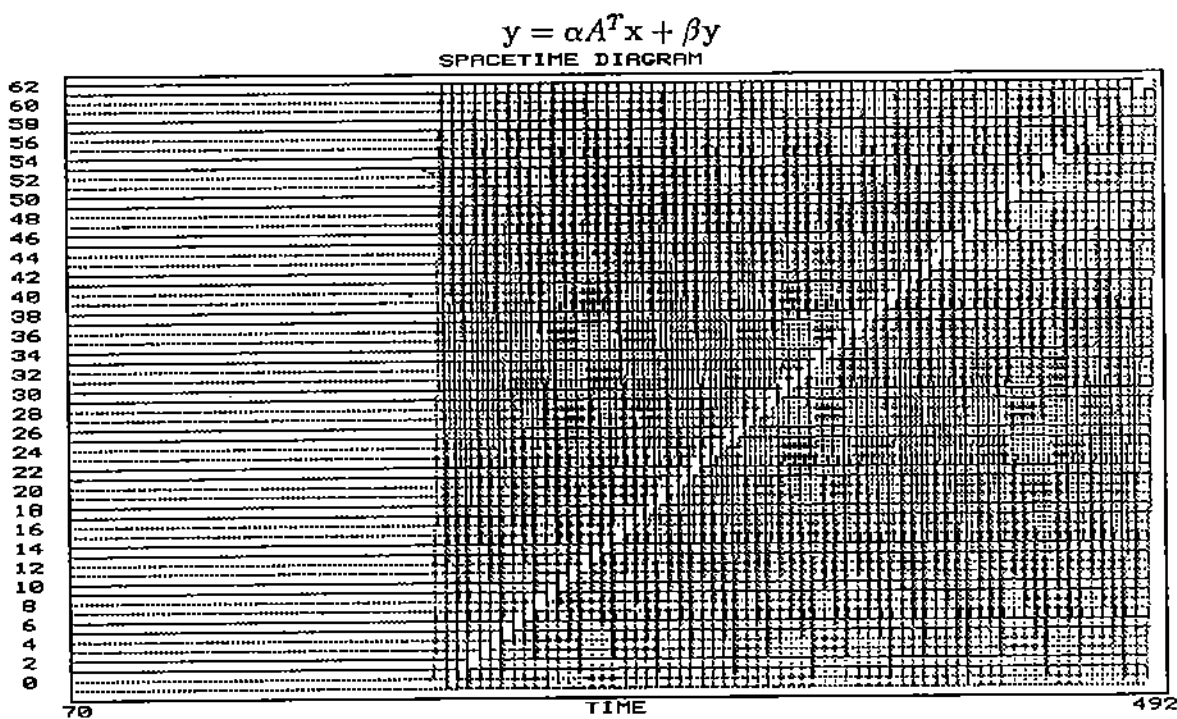


FIG. 8. Timespace diagrams on the nCUBE II for the data considered in Figure 6.



REMICN ROSSMCOOBY

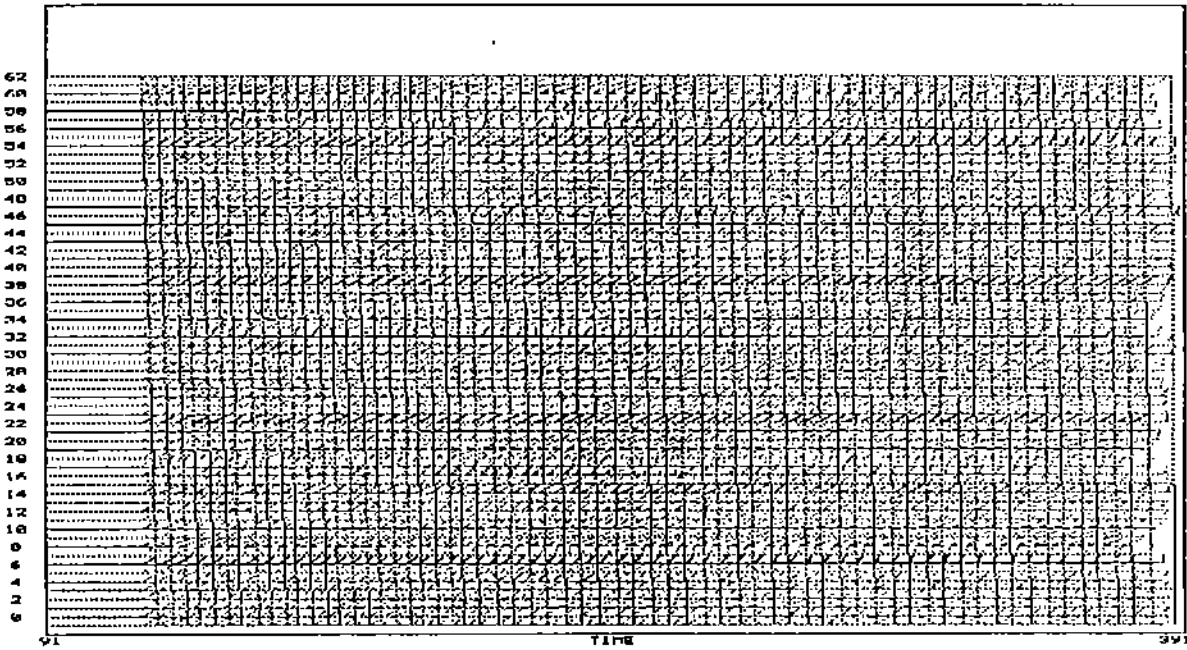


REMICN ROSSMCOOBY

FIG. 9. Timespace diagrams on the iPSC/860 for the data considered in Figure 7.

$$y = \alpha Ax + \beta y$$

SPACETIME DIAGRAM



$$y = \alpha A^T x + \beta y$$

SPACETIME DIAGRAM

