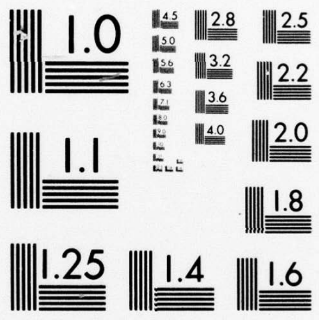

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

Parallel Computers – Parallel Mathematics, M. Feilmeier, (ed.)
© International Association for Mathematics and Computers in Simulation, 1977

NONLINEAR FILTERING ALGORITHMS FOR PARALLEL AND PIPELINE MACHINES[1]

R. S. Bucy[2] and K. D. Senne[3]

SUMMARY

The numerical resolutions of the important problem of the optimal processing of noisy observations to obtain optimal estimates of an underlying stochastic process or signal process is considered. The associated mathematical problem consists of the solution of a parabolic partial differential equation driven by the stochastic observations. The solution of this equation is the conditional density of signal given the observations. The purpose of this paper is to examine the impact of modern parallel and pipeline machines, such as CDC Star 100, Illiac IV, Cray 1, and AP120B Array Processor on the solution time and on the algorithmic program structure necessary to effectively use the capabilities of each machine. The mathematical problem suffers from the "curse of dimensionality"; For example, when the signal process has state dimension two, the partial differential equation has two spacial dimensions and taxes the capabilities of third generation serial machines. This is because our problem involves hundreds of solutions of the partial differential equation for different observation sequences, in order to determine error performance. With the advent of parallel and pipeline machines, higher state dimensional problems became feasible.

This paper will detail the mathematical statement of our problem and its general nature and importance in view of its relationship with partial differential equations. We will review the special architectural features of the various machines with emphasis on those which we can exploit for our problem. Next, we will describe the software algorithms developed for each machine and describe how the machine capabilities and limitations influenced the structure of the algorithm for each machine. The speed of the various machines will then be compared for our problem. Finally, we will describe the machine structure which we feel would be most effective for our problem both in terms of speed and also compatibility with natural algorithms for problems of higher space dimension.

1. INTRODUCTION

We will be interested in solving the following partial differential equation;

$$dp = Apdt + (h-\hat{h})'R^{-1}(dz-\hat{h}\,dt)p \qquad (1.0)$$

with $p: R^n \times R^+ \to R^+$, $h: R^n \to R^S$ and $\underline{z}$ a stochastic process taking values in R^S. The function $\hat{h}$ is the integral of h with respect to the conditional probability density p. In general, A can be any second order parabolic operator. Physically, A is the adjoint of the infinitesimal generator of the Markov diffusion vector process which represents the signal process, x. The observation process z is given by $dz = h(x)dt + d\nu$ with ν white noise with spectral matrix R, while p is conditional probability density of the signal at time t given the observations before time t. Details and background on how (1.0) arises can be found in [1]. In particular, (1.0) must be interpreted via the Stochastic Integral of Ito [2].

For numerical purposes it is convenient to consider a discrete version of the problem, where we replace the continuous observations by a discrete sequence sampled from the continuous process at a rate high enough to insure that the discrete problem is close to the continuous one. In this case (1.0) becomes

$$P(n+1) = S*F(n) \qquad (1.1)$$

$$F_{n+1} = 1/K_{n+1}\, D_{n+1}\cdot P_{n+1} \qquad (1.2)$$

The first equation, the convolution of S with F_n, is the density of the measure of signal process Δ seconds later if signal process had measure density F_n at time zero with Δ being the sampling rate. So P_{n+1} is the solution of (1.0) at time Δ if at time zero $p = F_n$ and R is infinite. Now (1.2) is approximately the solution of (1.2) at time $(n+1)\Delta$ if at time $n\Delta$ $p = P_{n+1}$ and $A = 0$. In terms of the discrete problem, $P_n\{F_n\}$ represent the condition density of x_n given $z_{n-1} \ldots z_0\{z_n, z_{n-1}, \ldots, z_0\}$, respectively. Of course, K_{n+1} is chosen so that F_{n+1} has total mass one. We note that the relationship of (1.0) to (1.1) and (1.2) is analogous to the Troller formula giving the semigroup with infinitesimal generator A+B in terms long products of alternate applications of the semi-group of A and that of B, i.e., formally,

$$\exp([A+B]n) = \exp(A)\exp(B)\ldots n \text{ times}\ldots\exp(A)\exp(B).$$

The representation (1.1) and (1.2) in fact has a desirable property not shared by direct differencing techniques applied to (1.0), namely solutions from non-negative initial conditions are <u>non-negative</u>. This property is of paramount importance for our problem as we seek a probability density solution of (1.0).

For the timing studies reported in this paper, a particular model was chosen;

[1]This research was supported at the University of Southern California in part by the United States Air Force, Air Force Systems Command, under Grant AFOSR-76-3100 and Contract ~~F44620-76-C-0005~~, and at the M.I.T. Lincoln Laboratory under Air Force Contract F19628-76-C-0002.

[2]University of Southern California, Los Angeles, California
[3]M.I.T. Lincoln Laboratory, Lexington, Massachusetts.

$$\underline{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix},\ h(x) = \begin{pmatrix} \cos x_1 \\ \sin x_1 \end{pmatrix},\ R = \begin{pmatrix} r_1 & o \\ o & r_1 \end{pmatrix},$$

$$A = q/2\ \partial^2/\partial x_2^2 + x_2\ \partial/\partial x_1 .$$

Further details concerning this problem can be found in [3]and [4]. We chose to examine the performance of the digital computers considered here in the context of this problem because of the wide experience we have had with this problem on a wide variety of machines. Further, because we used a machine independent (i.e., word-length independent) random number generator, see [5], we could directly compare computed numerical values on every machine. This was extremely useful for software development purposes. This random number generator produces statistically more reliable random numbers than those generators available as part of supplied scientific subroutine packages.

2. MACHINE ARCHITECTURE

The most common approach to speeding up the computations in serial machines involves the use of pipelines, or multistage processing. Pipelining involves the use of segmented functional units, with registers between segments, so that many identical functions can be overlapped in time at the maximum clocking rate, which is determined by the speed of the logic. Pipelining began to appear in many third generation machine architectures in the instruction fetch-decode-execute cycles. Recently, the pipelining concept has been extended to include arithmetic functions. The number of stages in an arithmetic pipeline is determined by the basic speed of the arithmetic function unit in relation to the memory minor-cycle time (the interval between successive fetches to memory). The combination of memory paging and pipelined arithmetic units has come to be known as a "vector processor." This terminology reflects the fact that optimal machine efficiency is obtained by streaming long vectors of operands through a single segmented arithmetic unit. The first implementation of this concept on a large scale was introduced in the CDC Star.

Although pipelining permits speed-up of serial operations to a maximum by the use of partial overlap, the results are not truly available in parallel. The array processor, on the other hand, contains a large number of identical functional units which may be actuated in a lock-step fashion to produce computations in parallel from different sets of data. Arrays may involve primative functional units, such as in array multipliers, or complete processing elements, such as employed in the Illiac-IV.

A. Parallel and Array Processors

Of the machines we consider, the Illiac-IV and Array Processor AP-120B are examples of differing parallel philosophy. The Illiac consists of 64 processing elements (P.E.'s) which act synchronously; each P.E. is in fact a C.P.U. with 2K words of memory. Each P.E. is theoretically capable of arithmetic speeds similar to a CDC-6600. Using Illiac at full capacity involves making every P.E. do something useful all the time. In full overlap mode, only recently achieved, see [3], Illiac can be doing memory fetches, for example, while the P.E.'s are operating. Illiac has a disk memory of 24 million 64-bit words, which must be exchanged with local memory that is limited to 2K words for each P.E.

The AP-120B Array Processor has an entirely different design philosophy. It is also a synchronous machine but the elements that operate in parallel have distinct functions: they are table memory, TM, data memory, MD, data pad x, DPX, data pad Y, DPY, S pad, SP, floating point adder, FA, floating point multiply, FM, and data bus, DB. The multiplier is a 3-place pipeline, the adder is a 2-place pipeline, while the memories are 3-place pipeline and 2-place pipeline for the data and table memory, respectively. The machine is synchronous with a 167-nanosecond cycle time. Data memory reads and writes can only be called every other cycle and reads and writes are finished in 3 cycles, while table memory reads and writes are accomplished in 2 cycles. The X and Y data pads are each 32 words and in one cycle reads from the last position of various pipes and writes to the other elements can be accomplished. Programs in the assembly language of the box, in order to be efficient, should attempt to keep all the elements operating on each cycle. The S pad does integer arithmetic to set memory read and write addresses, to do index computations, and to set memory pointers for DPX, DPY, SP and MD. The SP consists of 16 registers. Data words are 32-bit words. Effectively the box is capable of 12 million floating point operations per second. The major deficiency of the box is the date memory access time, which often tends to be the limiting factor in loop speed. Software development is time-consuming, although there are programs for the AP-120. The AP-120B is interfaced to a mini-computer and, because of the speed differential between the box and the host mini (30-40 times faster in the case of the PDP-11-55, for example), if speed is the object, very little computation can be done in the host. Reference [7] contains more details.

B. Vector Machines

The CDC Star-100 and the Cray 1 are examples of vector processors with 64-bit words. In the case of the Star-100 speed is obtained by streaming. A vector is located in consecutive memory locations and memory pulls put the consecutive memory locations into a pipeline in sequential order. For example, in add V1 to V2 the outputs of the two memory pipelines are inserted into the add pipeline. Note, the add pipe and the memory pipes operate concurrently with the write pipe part of the time, so that memory fetches and writes are not costly. The execution time for finding the vector sum is 100 + n/2 minor cycles where n is the vector dimension and the minor cycle time is 40 nanoseconds. The 100 cycles is set up time. Immediately two limitations are apparent to achieve maximum speed; n must be <u>large</u> compared to 100; and secondly, the components of the vector must be located in consecutive memory locations. The requirement of obtaining vector operands from sequential memory locations generates a significant data selection and reorganization problem. Data reorganization alone accounted for 86 percent of all identifiable overhead, which in turn represented 36 percent of the running time for our problem. Although a FORTRAN-based language is available for the Star, some of the more powerful instructions for merging and compressing vectors were only available by utilizing special in-line subroutine-like linkages to the assembly language.

The Cray 1 has a faster minor cycle time (12.5 nanoseconds) and the memory pipeline is connected to 64-word vector registers, which are in turn connected to the add pipeline, which outputs to another vector register. If the vector length is more than 64, its elements are processed by reading blocks of 64 into the eight vector registers. This architecture is responsible for a nominal set-up time much lower than that of the Star; so that the long vector requirement is ameliorated for the Cray 1. Speed is obtained by overlapping the memory to vector register, reads, writes, functional unit operation pipelines, as well as by synchronous operation of different function units when they have different input streams.

3. ALGORITHMS FOR PARALLEL NONLINEAR FILTERS

The algorithms for all the machines can be viewed in simple parts. Let us consider explicit forms of (1.1) and (1.2) as

$$P(n^{+1},x_i,\ y_j) = \sum_{k=1}^{N} S(y_j-y_k)F(n,[x_i-\Delta y_k],y_k) \quad (3.1)$$

where x_i i=1 ...M are subdivisions of $(-\pi,\pi)$ y_j j=1 ....N are subdivisions of $(-\pi/\Delta,\pi/\Delta)$ and where $F(n,[x_i-\Delta y_k],\ y_k) = \alpha F(n,x_H,y_k)+(1-\alpha)$ $F(n,x_L,y_k)$ and α is the interpolation constant, $x_L\{x_H\}$ is the nearest grid point to $[x_i-\Delta y_k] \equiv (x_i-\Delta y_k) \bmod 2\pi-\pi$ below {above}. The equation (1.1) takes the form (3.1) for the two dimensional phase demodulation problem, where the densities have been approximated by point masses of height $P(n+1,\ x_i,y_j)$ at point (x_i,y_j). Numerical solving (3.1) consists of three major parts;

Rearrangement: $F(n,x_i,y_j)\to F(n,x_L,y_j)$ (3.2)

Interpolation: $F(n,x_L,y_k)\to F(n,[x_i-\Delta y_k],y_k)$ (3.3)

Convolutation: $F(n,[x_i-\Delta y_k],y_k)\to P(n+1,x_i,y_k)$ (3.4)

Then, in order to complete one iteration, the explicit form of (1.2), namely

$$F(n+1,x_i,y_j)= \frac{D(n+1,x_i)}{K(n+1)}\cdot P(n+1,x_i,y_j) \quad (3.5)$$

must be solved. This operation leads to two other parts:

Data Update:

$P(n+1,x_i,v_j)\to D(n+1,x_i)\cdot P(n+1,x_i,v_j)$ (3.6)

Normalization:

$D(n+1,x_i)\cdot P(n+1,x_i,v_j)\to F(n+1,x_i,x_j)$ (3.7)

Explicit expressions for S and D can be found in [4], but they are not particularly useful here except for the fact they are <u>periodic</u> of period $2\pi/\Delta$ and 2π respectively, and $\overline{\text{F and P}}$ are also periodic in their first and second arguments of periods 2π and $2\pi/\Delta$ respectively. It is convenient to represent the matrix a_{ij} by a vector of $(M+1)N$ components such that:

$$\underline{V}(i+(M+1)(i-1)) = a_{ij} \quad i \le M \qquad (3.8)$$
$$= a_{ii} \quad i=\overline{M}+1$$

If $a_{ij} = F(n,x_i,x_j)$, then there is a permutation matrix P, so that

$$\underline{S} = P\ \underline{v} \qquad (3.9)$$

and S corresponds to $F(n,x_L,x_j)$ via (3.8), while the shift of S, k where k(i)=S(i+1), corresponds to $F(n,x_H,x_j)$. Now (3.9) accomplishes the rearrangement task. Interpolation can be accomplished by Star vector multiply, wherein the component of the product is the product of the components, i.e.,

$$I = S+W*(k-S) \qquad (3.10)$$

with W a vector of weights.

To prepare for the convolution the interpolated vector I reduced by removal of every M+1st element is periodically expanded as

$$E(i+M(j-1)) = I1(i+((j-1+N/2) \bmod N)*M)$$

for i=1 ....M , j=1 ....2N, N even, and I1 being the after surgery vector of every M+1st element or the last element each row in the matrix representation removed of I. This expansion eliminates the need for modular arithmetic in the evaluation of (3.1) when S is symmetric and has support contained within N grid points in y.

For Star the convoluation can be accomplished by the following sum:

$$\hat{S}(o)I1 + \hat{S}(1)(I1(-1)+I1(+1))+ \ldots$$
$$+\hat{S}(N/2)(I1(N/2) + I1(-N/2)) \qquad (3.11)$$

S(i) = S(y_i) and I1(k) being a subvector of E of dimension NM consisting of contiguous elements, the first element being 1+M(N/2+k)st element of E. Note that every element of I1 is called from memory N+1 times in forming (3.11). This method is not time consuming on Star because of the effective overlap in time of fetches from consecutive memory locations with the computation.

For both the Illiac and the AP-120B the convolution is performed by noting that for each fixed i, after F(n,·) is rearranged,(3.1) represents a one-dimensional convolution in k. So M one-dimensional convolutions are performed. Further, each element of the interpolated vector I1 is called from memory only <u>once</u> and each of $\hat{S}(i)$ i=1 ...N/2 is multiplied by it and each is accumulated to produce the convolution. This method of convolution was the basis for development of the Illiac-IV algorithm. In that case the N=128 elements of each row of F(n,·) is cyclically convolved with $S(\cdot)$ to produce a row of $P(n+1,x,\cdot)$. An adaptation of the same method for the AP-120B was suggested to us by Randy Cole of the U.S.C. Information Science Institute; the software for the AP-120B was developed by Jack Mallinkrodt of Communications Research.

One should note that our Star program represents the matrix as a vector of constructed columns. If one wished to do a convolution as above for Star, one would have to either read nonconsecutive memory locations or rearrange the interpolated vector to represent concatenated rows. In the former case, the memory pipeline would be inefficient because of the calls to non-contiguous memory locations and in the latter case the vector rearrangement would produce an equivalent time penalty. This situation reveals an example of the major architectural drawback of Star, namely, algorithms which use both column and row operation are not effective for Star. Note that (3.6) is also a row-oriented operation and as such will be accomplished differently on Star than on the other machines.

On the Star, (3.5) is accomplished by constructing a vector corresponding to $D(n+1,\cdot)$ with the first through Mth components the value of D and continued periodically, i.e.

$$d(i+M(j-1)) = \tilde{D}(n+1,x_i) \qquad (3.12)$$

$i = 1 \ldots M$, $j=1 \ldots N$. Now (3.6) is accomplished by a vector multiply of d by the vector resulting from (3.11). Normalization is accomplished by dividing the vector corresponding to (3.6) into two vectors of the same dimension and performing a vector add and repeating this is the number $K(n+1)$. This process is effective if $NM=2^L$; if this is not the case and a vector of odd dimension arises at any stage of the process, a zero is adjoined increasing the dimension by 1 and the process is continued.

In the case of the Cray each of the row vectors which make up $P(n,\cdot)$ are multiplied by the scalar $D(n+1,x_i)$ for the appropriate i. The AP-120B does the appropriate scalar version.

Because of the structure of Illiac with 64 independent P.E.'s, N=128 and M=32 were the grid sizes chosen for all problems in order not to penalize Illiac. The convolution and other row oriented operations were performed in two parts with all P.E.'s enabled. The rearrangement and interpolation were also used for the Illiac software.

4. EXPERIMENTAL RESULTS

A. Language Optimization

The code for the Star-100 was initially developed in Star FORTRAN, then each major piece of the program was timed with hardware timers and those pieces of code which were major contributors to the overall time were examined in assembly language. In particular, the rearrangement was originally accomplished with an indexed vector transfer instruction (vx to v), which had two vector arguments, the first, ℓ, corresponding to $F(n,x_i)$, y_j, and the second a vector of integers JNS. The value of vx to v was the vector corresponding to $F(n,x_\ell,y_j)$, i.e., $\ell(i)$ was obtained from F indexed by JNS(i). This instruction was found to be quite slow and an assembly language modification employing a more powerful version of vx to v, an indexed block transfer (of length M, for example), was used. Similarly, a vector sum instruction (sum), which has domain a vector and range the sum of its components, was replaced by code accomplishing normalization. Vector descriptors were also employed in the program. These optimization efforts produced code which executed 16.6 megaflops[4] or in other words, about 36 percent of the time was devoted to overhead operations rearranging, expanding, etc. In particular, Star ran five times faster than the 7600 on this problem, using the serial version of the Star software with the OPT=2 FTN level 410 compiler. Further, the performance of Star was exactly predictable from individual instruction time.

The Illiac-IV was coded in GLYPNIR and assembly listings were used to speed up the program. Our program, running in full overlap mode, where the array control unit (CU) can be operated simultaneously with arithmetic P.E. operations, achieved 9 megaflops. However, the Illiac is currently operating with a 80-nanosecond minor cycle time, rather than the design goal of 50 nanoseconds. Further, the actual time performance of our software suffers from inefficient language optimization, resulting from using the GLYPNIR language. It is expected that use of some assembly language in-line code or a higher-order language more sophisticated than GLYPNIR would produce slightly better running time. We will report more on this subject at a later date.

The AP-120B Array Processor was programmed in assembly language and the code seems perhaps within 10 percent of best possible code. Since the machine is synchronous, the code can be run on a digital simulator and theoretically timed with the result that it operated at 3.58 megaflops and executed arithmetic operations about 30% of the time. In the near future, we will have hardware benchmarks available. So far, we are attempting to benchmark the Cray 1 in the following way: First, a very fast serial FORTRAN program, developed on the basis of the Star code, will be used with the Cray 1 compiler to produce code. We have been assured that the Cray compiler will vectorize scalar code. Secondly, another FORTRAN program, which does the row-oriented convolution, will be tested. Finally, an assembly language version of our program will be prepared for the Cray 1. We hope to report on some of these timing studies at the meeting, potentially, the Cray could be 3 - 10 times faster than Star. On the solution of a linear system of equations the Cray 1 achieved 140 megaflops--see [10] and see Table 1 for direct machine comparisons. Finally, a 3-dimensional, combined amplitude-phase demodulator has been run on Star, with density represented by a 25K dimensional vector (see [3]).

[4] Megaflops = millions of floating point operations per second.

We hope to run the Cray 1 on the same problem and to compare running time.

5. REFERENCES

[1] R. S. Bucy, P. D. Joseph, "Filtering for stochastic processes with applications to guidance," Interscience, New York, 1968.

[2] K. Ito, "On stochastic differential equations," Memoirs Amer. Math. Soc., 4, 1951.

[3] R. S. Bucy, K. D. Senne, H. Youssef, "Parallel, pipeline and serial realization of optimal demodulators," Stochastic Control, Roxin and Sternberg, Editors, Marcel Delcker, New York, 1977.

[4] R. S. Bucy, C. Hecht, K. D. Senne, "An engineer's guide to building nonlinear filters," F. J. Seiler Lab. Report #SRL-TR-720004, United States Air Force Academy, Colorado Springs, Colorado, July 1972.

[5] K. D. Senne, "A machine independent random number generator," Stochastics, 1, 3, 3-23, 1973.

[6] System guide for the Illiac IV user, Institute for Advanced Computation, Ames Research Center, Moffett Field, Calif. 94035.

[7] AP-120B, Processor Handbook: Software development packages; floating point systems. 7259-02, Beaverton, Oregon.

[8] Cray-1 computer reference manual 220004. Cray Research, Inc., Bloomington, Min. 1976.

[9] Star reference manual. NASA Langley Research Center, Hampton, Va.

[10] D. A. Calahan, W. N. Joy, D. A. Orbits, "Preliminary report on results of matrix benchmarks on vector processors," Report #94, System Engineering Laboratory, University of Michigan, May 1976.

[11] R. S. Bucy, C. Hecht, K. D. Senne, "New methods--nonlinear filtering," Revue Franc. d'Automatique de Rechearche Operacionale, J-1, Feb.73, 3-54.

TABLE 1

PERFORMANCE OF VECTOR PROCESSORS ON THE PHASE MODULATION PROBLEM

Mega-flops/ Dollar	Machine	Mega-flops/ Theor.	Mega-flops Actual	Time per Iteration
-	Illiac4	64	9	9 millisec.
3-5	Cray 1	80	-	-
2-4	Star-100	50	16.6	4.9 msec.
16-32	AP-120B	12	3.57	22.7 msec.
-	6600	2	.630	130 msec.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

18 19

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: AFOSR-TR- 78 - 0649
2. GOVT ACCESSION NO.:
3. RECIPIENT'S CATALOG NUMBER:

4. TITLE (and Subtitle): 6 NONLINEAR FILTERING ALGORITHMS FOR PARALLEL AND PIPELINE MACHINES.
5. TYPE OF REPORT & PERIOD COVERED: 9 Interim rept.,
6. PERFORMING ORG. REPORT NUMBER:

7. AUTHOR(s): 10 R. S. Buc~~k~~y [redacted] K. D. Senne
8. CONTRACT OR GRANT NUMBER(s): 15 AFOSR-76-3100

9. PERFORMING ORGANIZATION NAME AND ADDRESS:
University of Southern California
Department of Aerospace Engineering
Los Angeles, CA 90007

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 16 17
61102F 2304/A1

11. CONTROLLING OFFICE NAME AND ADDRESS:
Air Force Office of Scientific Rsch/NM
Bolling AFB, DC 20332

12. REPORT DATE: 11 1977
13. NUMBER OF PAGES: 5 12 7p.

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office):

15. SECURITY CLASS. (of this report): UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

PROCEEDINGS OF THE IMACS(AICA)-GI, SYMPOSIUM, March 14-16, 1977
Technical University of Munich, PARALLEL COMPUTERS-PARALLEL MATHEMATICS,
M. Feilmier Editor, pp 93-97.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

nonlinear filtering
phase demodulation
megaflops
software structure
AP120B
Star 100

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Results are given for the timing of various supercomputers for the problem of phase demodulation. The effect of computer architecture on software development is detailed.

402079

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)