A Technique for Real-Time Data Preprocessing

Mario R. Schaffner

Reprinted from

Symposium on Machine Processing of Remotely Sensed Data

June 21 - 23, 1977

The Laboratory for Applications of
Remote Sensing

Purdue University
West Lafayette
Indiana

IEEE Catalog No.
77CH1218-7 MPRSD

A TECHNIQUE FOR REAL-TIME DATA PREPROCESSING

MARIO R. SCHAFFNER
Massachusetts Institute of Technology

ABSTRACT

A processing system is presented that implements simultaneously the efficiency of the special-purpose processor and the total applicability of the general-purpose computer — characteristics commonly though of as being mutually exclusive. The solution adopted is that of specializing the machine by programming the hardware structure, rather than by adding software systems to it. Data are organized in circulating pages which form a plurality of local dynamic memories for each process. Programs are made up of modules, each describing a transient special-purpose machine. Applications to real-time processing of radar signals are referred to.

I. INTRODUCTION

Remotely sensed data, in raw form, often constitute an exceedingly large and redundant data set with respect to the application needs. Therefore, on-site preprocessing would be highly desirable in order to transmit, or to record, a smaller amount of data. Because characteristics of the raw data might be crucial for some applications, this preprocessing needs not to be a mere data reduction process, with undesirable loss or degradation of information, but rather it should produce appropriate data transformations that, while reducing the amount of data, keep the relevant information undegraded for the largest number of applications. It should be pointed out that there is no unique way to specify such transformations, and continuing development is expected in this field.

For many applications which require quantitative data, or in which calibration and stability are of concern, digital processing becomes mandatory. Sometimes, equipment size and power are also relevant. In these cases, special digital processors constitute an efficient solution, yet this solution necessitates new design and equipment whenever a different preprocessing is required. Today computer availability indicates as a more appropriate solution the use of general-purpose computers, with specialization obtained through software.

However, many complex real-time processings needed in this context are not possible by means of affordable general-purpose computers, a fact which in turn makes the expectation of generality deceptive. Moreover, the development of software systems constitutes sometimes a large task in itself. Also, frequent processings are in the class of pattern recognition, which typically requires a large degree of parallelism, and which often necessitates a variety of strategies that can be formulated only with difficulty in a single programming language.

Notwithstanding what one might think at first, in this paper we question whether the efficiency of the special-purpose processor and the generality of the general-purpose computer are necessarely mutually exclusive. In the context specified above, we present a solution to this problem of conflicting goals: specializing the machine by programming the hardware structure, rather than by adding software systems to it. The desirability for such an approach has certainly arised in several contexts; in some sense, analog computers were taking the same direction. What apparently has impeded the development of this approach is the lack of a language suited for this task; that is, a language that permits both (1) an effective and efficient representation of the processes in terms of computational structures and data structures, and (2) a direct implementation of these structures by means of suitable hardware.

In the next section, a language which has these two characteristics is presented. In section III more details are given on the corresponding hardware, and in section IV aspects of the programming are discussed. In section V, an example is given of a preprocessing in a specific field of application. References are referred to for more extended descriptions of this processing approach, and of the applications made.

II. THE LANGUAGE

A. OUTLINE

This language departs from the typical form of commands and declarations, connected with a phrase syntax. It has instead the form of abstractions of special-purpose machines that one might wish to create for the various tasks.

There are two basic modules: (a) data blocks, called pages, each of which contains the data pertaining to a process or machine; (b) program blocks, called description of structure (DS), or states, which describe instantaneous special-purpose machines.

Processing is obtained by appropriate unions of blocks of one type with blocks of the other type. This gives a frame for modeling parallel activities, independent or concurrent. Also, it permits to implement different strategies by means of machines specifically designed, and thus efficient.

B. THE BASIC FRAME

To simplify the description of special-purpose machines, a common basic frame is assumed. This frame refers to activities which are common to all processes, such as input, output, data transformation, and storage; however, no rigid constraints are imposed. The frame has the form of the pipeline general structure of Fig. 1, comprising several stations through which data blocks (the pages) circulate. In one station, the assembler, new data can be acquired; accordingly, the input sources are connected to the assembler. In another station, called programmable network or PN, data can be transformed. In a station called packer, data can be routed elsewhere; accordingly, the output devices are connected to the packer. In the page memory, the data blocks rest, in accordance to the configuration given to this memory.

C. THE PAGES

The working data set of a process, that is, the data presently used by a process, or a portion of a large process, forms a data block which moves as a whole through the register arrays Ω of the frame indicated in Fig. 1. Such a data block is called a page. The pages are generated, work, and disappear as described in the sequel. The role of these pages will be commented in subsection G.

D. DESCRIPTION OF STRUCTURE (DS)

To perform a specific process, a specific structure has to be given to the frame of Fig. 1. The description of this structuring of the frame is divided in the following four components.

1. Input prescription I. In the assembler, specific connections with the input sources have to be activated if new data have to be transferred into the page presently in the page array Ω_a. The prescription of such data acquisition, in some code, is indicated globally with the symbol I. Beside the identification of outside sources, prescription I may also give numerical constants as new input data.

2. Data transformation F. If the working set of a process (the page) is simultaneously available in the register array Ω_N of PN, it is no longer necessary to decompose the process into a sequence of instructions (instruction = opcode + addresses). It is more appropriate to view the process as a (minimal) set of global transformations for the page; by global transformation is meant here what can be done at any given time with the data presently in the page. To implement such global data transformations, a network of processors which can assume a variety of operational configurations is used. For this reason, the station where data transformations are performed is called programmable network, or PN. We indicate globally with the symbol F the code needed by PN for implementing such data transformations.

3. Transition function T. When we delineate a page transformation as a part of a process, it is also necessary to indicate which other transformation should be next for the page. Given the level of activity performed by a page in one circulation, a predetermined succession of Fs (like a sequence of instructions) will not be the typical occurrence. It is more convenient to consider after each F a transition function for establishing the transfer to one of several other data transformation F, in response to results obtained in the page, or to outside signals. The same programmable network is used for implementing this transition function, immediately after the execution of a data transformation F. We indicate globally with the symbol T the code needed by PN for performing such a transition function.

4. Routing R. In the packer, specific connections have to be activated if specific data in the page, presently in the array Ω_p, are routed to specific output devices. The prescription of such routings, in some code, is indicated here globally with the symbol R. Beside routing data elsewhere, prescription R may also indicate the erasing of data in the page. Routing can be made transition dependent, in the sense that different actions are performed depending on the outcome of the transition function T.

From the above, we have arrived to the description of a complete processing structure symbolized in the form of a quadruplet [IFTR]. It includes input connections in the assembler (component I), data transformations and transition functions in the programmable network (components F and T), and output connections in the packer (component R). We call such a quadruplet a description of structure, or DS. The structure described by a DS is an instantaneous structure, and to implement a process we generally need several such DSs.

E. STRUCTURE AND DATA MACHINES (SDM)

In the frame of Fig. 1, a processing activity is implemented by relating a page (a data set) to a quadruplet (description of structure). In general,

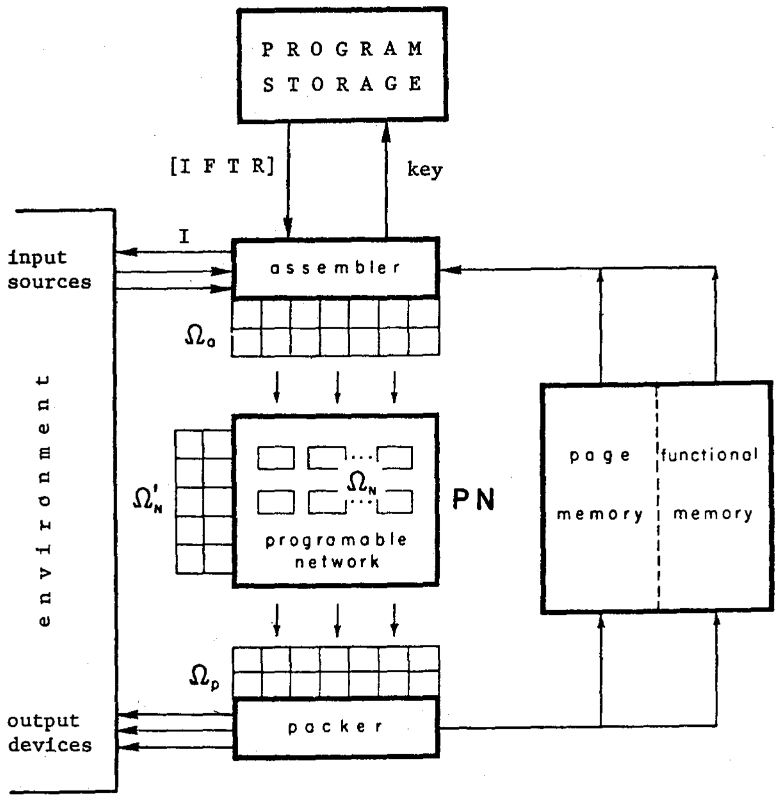


Figure 1 - The basic frame for describing processes

there will be many pages circulating in the frame, and many quadruplets in a program storage (Fig. 1). To implement the pairing of pages with quadruplets, the pages carry a word called the key. The key is given the label of a quadruplet existing in the program storage. Everytime a page enters the assembler, it acquires a quadruplet in response to the content of its key. Component I of the quadruplet is used immediately in the assembler for acquiring new data. Component F is used for transforming data as soon as the page is transferred into PN. Component T, also used in PN, may change the content of the key, so that at the next circulation the page may acquire a different quadruplet. Component R is used for routing data when the page is transferred into the packer. In this way, each page, at each circulation, implements a particular processing activity.

As said above, each pairing of a page with a quadruplet is determined by a previous outcome of the transition function T; this outcome depends on the results in the page, which in general depend also on the input data. This mechanization makes it possible to organize large varieties of activities which self-develop and change in time. In order to have a criterion for distinguishing these activities, we use the notion of Structure-and-Data Machines, or SDM; each SDM being implemented by a page (the data) through a pattern of pairing with DSs (the structure described by the quadruplets). Denoting specific patterns through DSs by indexed brackets, we can write symbolically

$$\text{page}_n + [\text{DS}]_m \longrightarrow \text{SDM}_k$$

To create a page, it is sufficient to insert a key into circulation; this can be implemented either as an initial action of the operator, or as routings by already existing pages. A particular outcome of the transition function makes a page to disappear. The same DSs can be shared by many pages; a page can go, at different times, through different paths of DSs. Each SDM is equivalent to a special-purpose machine; a large number of such machines can simultaneously execute processing on the same or different data in the frame of Fig. 1.

F. FACILITIES

the SDMs outlined above are transient machines, created at specific moments for performing single tasks. To make the work of the SDMs concurrent toward larger tasks, several facilities are provided in the frame of Fig. 1.

Auxiliary page array. Exchange of data among pages is a requirement for a cooperative work of SDMs. For this purpose, PN is provided with an auxiliary page array Ω'_N ; when a page is in PN, it can transfer some of its data into Ω'_N ; some other pages, when in PN, can acquire or exchange those data from Ω'_N . The data in Ω'_N are not removed during the flowing of the pages in PN.

Driven transitions. Also messages can be stored in Ω'_N ; a very useful feature is the transmission of a key, through Ω'_N , by part of one page to specific other pages; in these pages, the new key overrides the outcome of their transition functions. In this way, it is possible for a page to direct the behavior of other pages. A transition so produced will be called a driven transition.

Page memory. The pages are stored by the packer into the page memory as blocks of data. The assembler acquires pages from the page memory as blocks of data. The input-output discipline of the page memory is an issue which pertains to the work of the entire system, rather than to the single SDMs. If there is no relation among the work of the several pages, or if this work requires a simple sequential order of the pages, the simplest structure for the page memory is that of a FIFO discipline; that is, the assembler automatically receives one page after the other, in the same order in which the packer produced those pages. This is the configuration of the page memory for the applications described in section V. Different classes of problems may require different structures of the page memory; these can be activated when needed by means of appropriate codes routed to the memory control.

Functional memory. In modeling complex processes, a storage common to all pages is often appropriate. However, in most cases in which data are stored outside the page, some simple operation is actually required, such as accumulating a sequence of data, storing the maximum (or minimum) value in a sequence of data, or counting the occurrences of a set of values (e.g., to produce a distribution). For this purpose, a functional memory is provided in the frame of Fig. 1. Every SDM can route some data of its page into specific locations of the functional memory, and the routing prescription specifies the function with which the data should be acquired. The functional memory provides for the execution of these functions, thus relieving the SDMs of many clerical tasks.

G. THE MODELING OF PROCESSES

A process is modeled as the product of the activity of SDMs. In the simplest case, an SDM implemented by one page and one DS may suffice. In general, a large number of SDMs, implemented by pluralities of pages and DSs will be required.

The agents of the activity are the pages. Pages are created when specific tasks are needed; with circulation through the structure of Fig. 1, the pages keep their data working sets updated; data can be acquired from input sources, from other pages, from the functional memory; data can be routed to output devices, to other pages, to the functional memory; the pages provide for the computation; any page can take control of other pages; when its task is accomplished, a page normally disappears. The outcome of the page activity may be found in the forms of results in output devices, intermediate data in the functional memory, and newly generated pages in circulation.

All such activities are accomplished by the pages in accordance to the SDMs that happen to be implemented. The SDMs are implicitely described in the DSs, but what actually takes place may strongly depend on the input data and consequent results. These different results may produce different use of the DSs, often iteratively in time or in space (arrays of pages), and sometimes recursively. The main mechanisms for building these dependencies are the transition functions, the transition-dependent routings, and the self-generation of pages.

As indicated in the introduction, this language applies equally well to the abstract description of processes (in a structural form), and to the description of configurations to be assumed by a physical machine for implementing the described processes. In the next two sections, some points are discussed on these two phases.

III. COMMENTS ABOUT IMPLEMENTATION

The frame of Fig. 1 is conceived in a space-time domain; therefore, it is suitable for direct physical implementation, for instance, with digital technique. In effect, Fig. 1 represents an architecture for a computer, and the quadruplets described in section II constitute the machine language for such a computer. The main characteristics of this architecture are the organization of data in pages constituting working sets for the processes, their automatic circulation through a pipeline series of stations, and the use of structure descriptions for specializing these stations by the pages themselves; accordingly, a physical implementation working in this way will be called a Circulating Page and Structure (CPS) machine.

The implementation of a CPS machine is discussed elsewhere.[1,3] This work started from the need of processing radar signals in real time, in ways that could easily be changed to follow developing research. After several special processors based on circulating words related to independent processes, at the Radio Meteor Project of the Smithsonian Astrophysical Observatory, Cambridge, Mass., and at the Weather Radar Project of the Massachusetts Institute of Technology, the first general-purpose machine of this type, called CPL 1, was constructed around 1969, and put in operation at the Smithsonian meteor radar station in Havana, Ill. in 1970.[2] Then, the machine was brought to MIT and used for experiments of real-time characterizations of weather radar echoes.[4] From these experiences, the design and the construction of a second machine has started, and it now proceeds at the National Center for Atmospheric Research, Boulder, Col.

Only a few comments are made here from the viewpoint of processing in real time large quantities of data. The pipeline structure of Fig. 1 leads to an orderly organization of the processing for a continuous flow of data from input sources to output devices. Because of the allocation of data in circulating pages, there is basically no use of addresses, a fact which facilitates a fast execution as needed for real-time processing.

It can be noted that this modularity of data blocks and program blocks makes multiprogramming feasible without overhead. Each transfer of a page can be viewed as an interrupt, in which the status of the process is automatically held by the key of the page. Each page may very well belong to a different program; each program can be implemented by many pages. In either case, there is no need of scheduling systems, with consequent overhead in execution time and program complexity. These benefits, as well as several others, derive from the fact that the processing is page driven.

IV. COMMENTS ABOUT PROGRAMMING

In the language of section II, a process is completely described by the DSs. For instance, a process may be implemented by one or more pages pairing sequentially with DS_1, DS_2, DS_2, DS_3, DS_2, DS_2, DS_4; for such a process, the program consists simply of the set DS_1, DS_2, DS_3, DS_4, being the sequence of pairing indicated by the components T in the DSs, and the number of pages by their components R.

These programs are modular in terms of DSs; every program consists of sets of DSs related to each other by transition functions. Such a structure is operatively equivalent to that of the finite-state machines defined in automata theory; for this reason, we will refer to the DSs also as states, and we will view the programs as state diagrams. The state structure of these programs can have utilizations similar to those of finite-state machines, such as recognition of patterns, or memorization of past events. At the same time, the text within the states (components I, F, and R of the quadruplets) permits to handle a continuous flow of data, and to produce computation on that data.

Because of all the above, the representation of these programs is in a particular form of state diagram. States are indicated as encircled domains (see Fig. 3). The data transformation F and the transition function T are indicated inside these domains, above and below a horizontal line, respectively. The outcomes of the transition functions are indicated by arrows connecting the states. The input prescriptions I appear implicitely as input data in the expressions of F and T. Routings are indicated outside the encircled domains, along the paths followed by the page.

The state diagram is the medium used by the user for constructing a program. At the end of the programming work, the elements of the resulting state diagram are expressed in corresponding codes of the CPS machine available. Because the work of a CPS machine follows the same mechanization expressed in the state diagram, the production of these codes represents a simple clerical task.

In this situation, the notations used in the state diagram can be completely arbitrary. In Fig. 2, as an example, some working notations are shown which will be used in the program described in section V. A full set of notations is described in.[3]

In the software domain, we can distinguish two sectors: the programming language, which provides the user with the facilities for prescribing specific processes; and the packages of code, to which various activities relate, such as debugging, system management, and program maintenance. In the CPS approach, these two sectors are not separate to the extent as they are conventionally, because of the identity of structure between the user model of a process and the actual work of the machine. Here a package of code is basically a particular transliteration of the items composing the state diagram developed by the user. It is a situation similar to that of a program written in assembly language, except that the structure of an assembly program is quite far from (often it has little to do with) what the user had originally in mind. This similarity between the user model and the machine execution is particularly helpful in the phases of debugging and modification or extension of programs.

There are two forms for describing parallelism in this language. One is of simultaneous execution of operations in different variables of the page; an example is in state 4 of the program in Fig. 3. The other form is a virtual parallelism produced by a plurality of pages pairing with the same DSs; the program in Fig. 3 makes use of this parallelism also.

V. EXAMPLE OF REAL-TIME DATA PREPROCESSING

As an example of application of this processing approach, a simple program is described which has been used with the CPL 1 machine in experiments of real-time preprocessing of weather radar echoes.

The context is monitoring a storm activity within the radar range. Two types of information are of interest, the total precipitation and the trend of the storm. Estimates of precipitation can be derived from point-by-point values of the radar echoes; the trend can be inferred from appropriate characterizations on the echo patterns. The echo values need to be measured everywhere and continuously, whereas the characterizations need to be determined only at periodical intervals of time. Methods of characterization have been described elsewhere;[3, 4] here a program for the recording of distributions of echo intensity is described in detail.

Weather echoes are highly fluctuating because of the motion of the water droplets; therefore, several (e.g., 32) radar returns need to be averaged at each point in order to obtain meaningful measurements. Moreover, because these measurements are collected automatically without screening by operators, non-weather echoes, such as echoes from the ground, either in line of sight or through anomalous propagation, should automatically not be accounted.

To discriminate ground echoes, a method is adopted [4] which is based on the different fluctuation exibited by weather and ground echoes.

To facilitate the data analysis, the measurements are collected in the form of areas covered by echoes within different levels of intensity (e.g., 15) every two minutes.

To accomplish all these functions, the system of SDMs represented by the state diagram of Fig. 3 is used. There is one SDM, implemented by one page and states 1-3, which controls the entire activity; and a plurality of SDMs, implemented by a sequence of pages (one per range point) which follow states 4 and 5, for performing the computation. All pages circulate in synchronism with the pulse repetition frequency of the radar. First the activity of the control page is described, and then that of the computation pages.

To initiate the process, a page is inserted into circulation, in state 1. In this state, variable A acquires continuously the current azimuth az of the radar antenna. At each 2-minute signal t from a digital clock, the control page transfers from state 1 to state 2; at this transfer, by routing, a sequence of computation pages is produced, and variables M, N, Q, and the distribution in the functional memory are cleared.

In state 2, the page remains idle for 31 circulations, and then routes a driven transition to state 5 for the following pages. When the current azimuth becomes equal to variable A (which occurs at the completion of an antenna rotation), the page transfers to state 3, and routes a driven transition to state 0 (disappearing) for the other pages.

In state 3, variables A, B, and C of the page acquire the values in M, N, and Q, respectively, of the functional memory; these values are normalized by means of given constants a, b, and c, and then are routed to the output. Also a command for transferring the content of the distribution to the output is produced. The page returns to state 1, except if a stop signal is present, in which case also the control page disappears.

The computation pages are generated in state 4. In this state, variable A computes at each circulation the absolute value of the difference between two consecutive digital samples s of the radar echo at the range point to which the page belongs. Variable B accumulates in time these differences, and variable C accumulates the values of the samples s.

Every 32 circulations, these pages are driven to state 5 by the control page. In state 5, C is divided by 32, so that it represents a mean echo value. Then the page returns to state 4, but following different paths, and thus producing different routings, in dependence on the results obtained. If the mean echo (variable C) is above a threshold h (assumed above the noise value), and the accumulation of the differences (variable B) also is above a threshold g (chosen for optimum ground-echo discrimination), the echo value is routed to a distribution

Capital letters

A , B , C , ···	denote variables in the page

Capital letters

M , N , Q , ···	denote variables in the functional memory

Lower case letters

a , b , az , ···	denote input data

IN THE DATA TRANSFORMATION

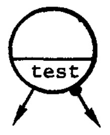
Parallel operations

A B C $\sum$ 2 A s

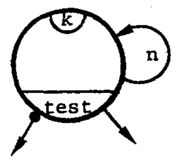
$A(i+1) \leftarrow A(i) + 2$
$B(i+1) \leftarrow B(i) + A(i)$
$C(i+1) \leftarrow C(i) + s$

i denoting values before operation, and i+1 after

IN THE TRANSITION FUNCTION



dot indicates the path followed when test is true.



the page circulates n times in state k, before following the transition function described in k

or state 0 — the page disappears (after routing executed)

IN THE ROUTING

A B	variables A and B are routed to output
A_o B_o	variables A and B are cleared
$A \rightarrow M(\Sigma)$	variable A in the page is routed to variable M in the functional memory with function accumulation ($M \leftarrow M + A$)
$C \rightarrow$ distr	in the functional memory a variable labeled with the value of C is incremented by one
N(incr)	variable N in the functional memory is incremented by one
drt k	driven transition to state k for the following pages
n Pag(k)	n pages are generated in state k

Figure 2 - Notations used in the state diagrams

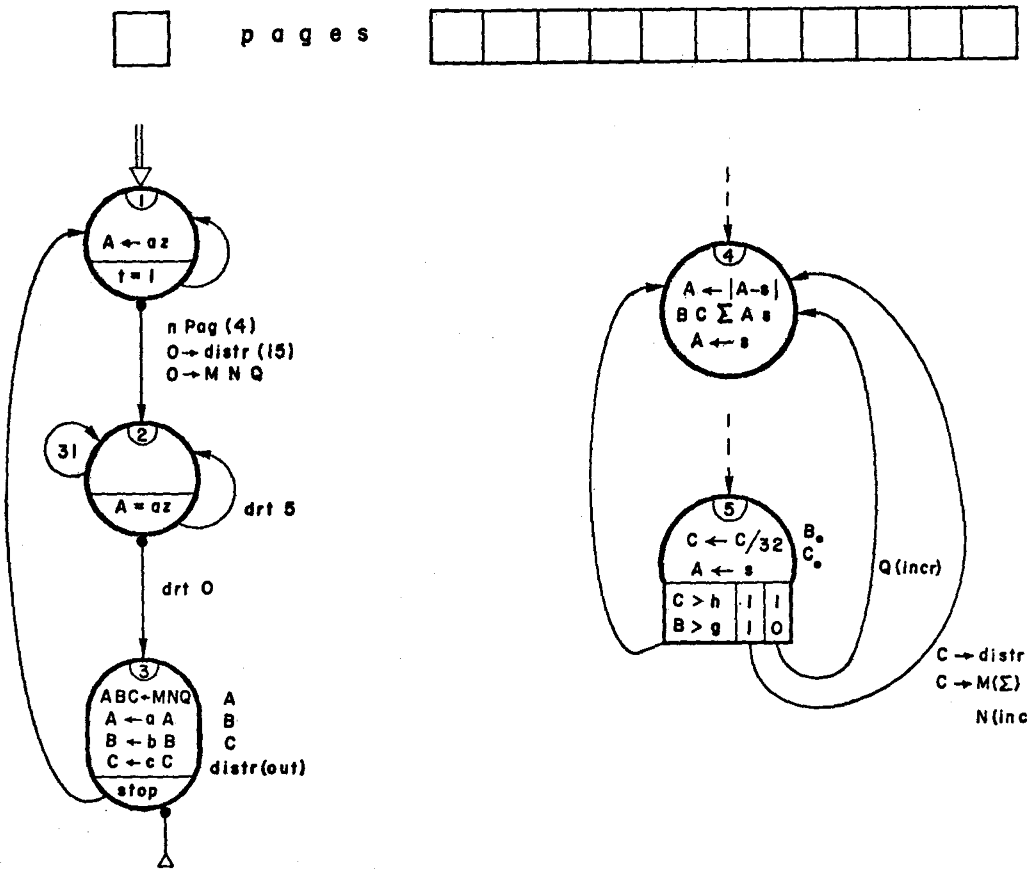


Figure 3 - State diagram of a real-time preprocessing of radar signals

function, and to an accumulator M, both in the functional memory; also an increment is routed to variable N in the functional memory. If the echo value in C is above the threshold, but the accumulation B of the differences is below the threshold g, that is, the echo very likely is not from weather targets, an increment is routed to variable Q in the functional memory. If both C and B are below their respective thresholds, the page transfers to state 4 without routings to the functional memory. In all cases, variables B and C are cleared before the page reaches state 4 (routing indicated beside the state circle).

In this way, variables M, N, Q, and the distribution in the functional memory gradually build up global results. The computation pages do the work at each point; the functional memory assembles the data; and the control page provides for the synchronization and the recording of the final results. The data produced are as follows. From variable N, an approximated measurement is obtained of the total area covered by precipitation. From variable M, a quantity is obtained which can be related to the magnitude of the storm; it is the area weighed by the echo intensity at each point; it will be referred to as the integral. Variable Q gives the total area covered by echoes recognized as not pertinent to weather targets. From the distribution, the areas with echoes within given intensity intervals are obtained.

The state diagram of Fig. 3 gives an easy visualization of the entire process, and at the same time provides all the elements necessary for the execution of the process by a CPS machine. The actual codes for the CPL 1 machine differ slightly from the elements of the state diagram of Fig. 3; nevertheless, they are contained in two punch cards.

Given the readability of a program in the form of a state diagram, and its direct correspondence with the machine activity, modifications and extensions of a process are tasks easy to actuate and to check. These are practical consequences of the modularity in data blocks (the pages) and program blocks (the states) commented in sections III and IV. Some comparison of program complexity and execution time with respect to conventional programs are in.[2,3]

In Fig. 4, an example of data obtained with this type of preprocessing is given. It refers to a storm which moved from the northwest to the southeast of Boston during the night of December 8, 1974, and which changed from snow to rain. The upper part of the figure plots the data recorded with the program of Fig. 3, running automatically in absence of personnel. The lower part of the figure reports some of the real-time characterizations made by an operator during interruptions A, B, C, and D of the automatic data collection.

The general trend of the storm is shown by the total covered area and the integral curves. the curves plotting the area of the single level intensities are of interest for analyzing the phases of the storm. In the development phase (e.g., at 17.30), areas of strong precipitation exceed areas of weak precipitation. In the decay phase (after midnight), the areas of the different levels form a regular monotonic sequence.

The characterizations consisted of measuring in selected regions the area A covered by precipitation, the mean intensity M in this area, and the mean echo gradient G of this echo pattern. Of particular interest is the fact that, from the value of the gradient, inference can be made on the type of precipitation, such as snow, convective shower, and stratiform rain.[4] The numbers which appear in the lower part of Fig. 4, obtained by processing radar signals, show a progressive increase in the value of G during the life and the movement of the storm. From observations made during two winters, this fact could be interpreted as a change of the precipitation from the form of snow to that of rain. The actual changes that occurred in this storm were in accordance with the inferences from these measurements.

Also these characterizations can be automated and included in a single, comprehensive monitoring program, on the line of the small one here described. For defining appropriate sets of parameters to be derived from the raw radar signals, an extensive analysis is necessary on the correlations between characterizations that can be made on the echo patterns and relevant characteristics of the associated meteorological events.

VI. CONCLUSION

The efficiency of the special-purpose processor and the flexibility of the general-purpose computer are not mutually exclusive. When these two characteristics are both simultaneously available, large quantities of data can be processed efficiently to produce small amounts of data with a high information content; and the processes can be readily transformed to follow changes in the requirements or developments in the understanding of the related physical phenomena.

ACKNOWLEDGEMENT

I wish to thank Frank Marks for the help in taking the measurements, and for the several discussions. This work has been supported, at different times, by the National Aeronautics and Space Administration under contracts NASr-158, NSR-09-015-033, and NASW-2276.

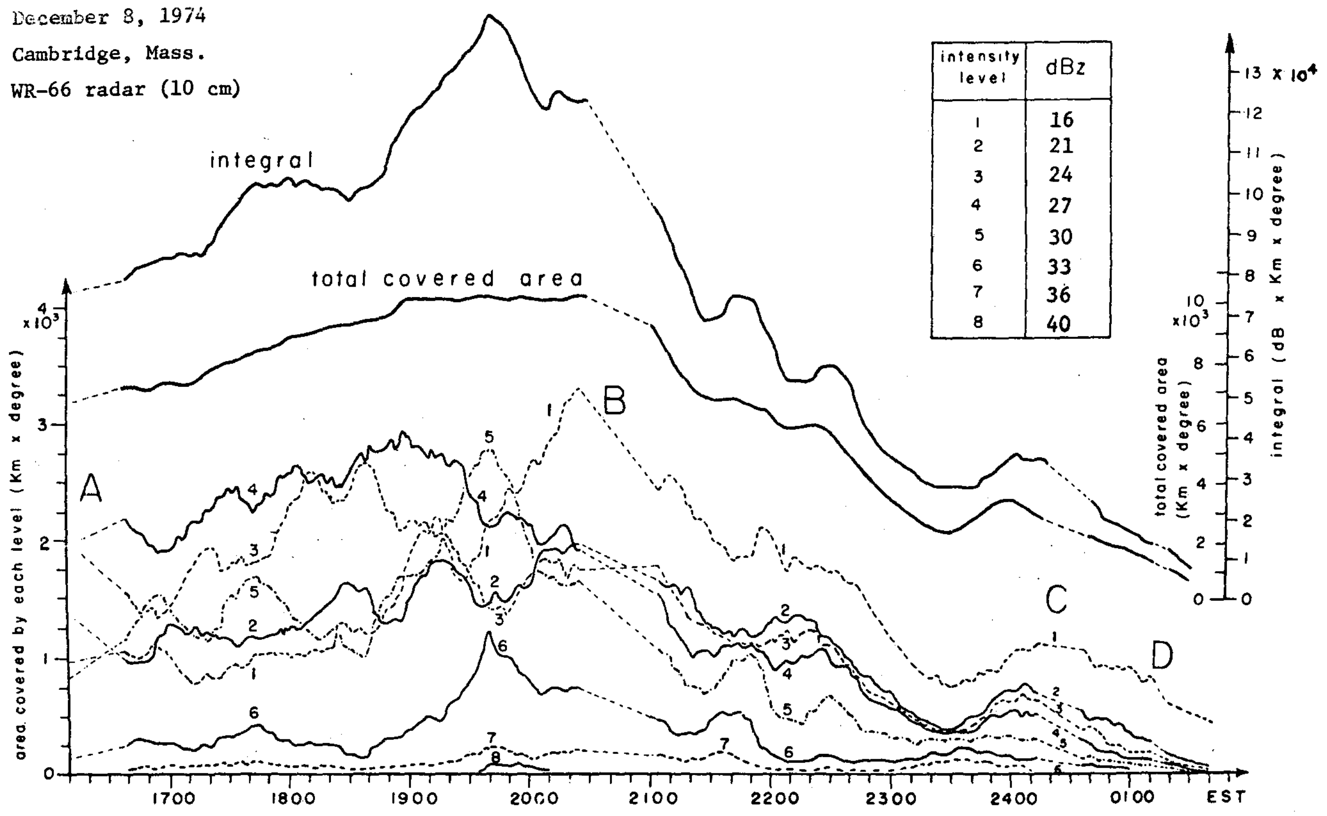
REFERENCES

1. Schaffner M R (1971), "A Computer Modeled after an Automaton", in Computers and Automata, Polytechnic Press, Brooklyn, N.Y., pp 635-650.

2. --- (1972), "Computers Formed by the Problems rather than Problems Deformed by the Computers", COMCON Dig., pp 259-264.

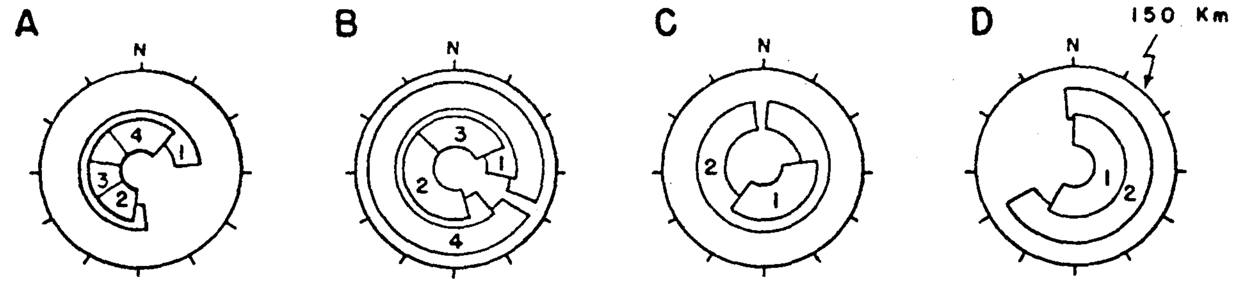
3. --- (1974), "Research-Study of a Self-Organizing Computer", Final Report, contract NASW-2276, NASA.

4. --- (1976), "On the Characterization of Weather Radar Echoes", Prepr. 17th Radar Meteor. Conf., Amer. Meteorol. Soc., Boston, Mass., pp 474-485.



intensity level	dBz
1	16
2	21
3	24
4	27
5	30
6	33
7	36
8	40

Samples of the analyses made during the interruptions



1 total	A = 6000	1	A = 6800	1	A = 2370	1	A = 1320
	M = 21.5		M = 23		M = 22.5		M = 20
	G = 5.5		G = 5.2		G = 7.9		G = 9.5
2 southern edge	A = 1400	2	A = 1730	2	A = 2600	2	A = 1730
	M = 19		M = 22.5		M = 22		M = 23.5
	G = 6.3		G = 8.6		G = 8.3		G = 9.3
3 center of the storm	A = 1830	3	A = 3630				
	M = 21.5		M = 25				
	G = 4.7		G = 3.8				
4 northen edge	A = 830	4	A = 2590				
	M = 19		M = 24				
	G = 7.8		G = 7				

A = covered area (Km x degree)

M = mean intensity (dBz)

G = mean gradient (dB/Km)

Figure 4 – Automatic (unattended) monitoring of a storm that crossed Massachusetts during the night of December 8-9, 1974, with sporadic interruptions by an operator for particular characterizations.