A production implementation of an associative array processor—STARAN

by JACK A. RUDOLPH

Goodyear Aerospace Corporation
Akron, Ohio

INTRODUCTION

The associative or content-addressed memory has been an attractive concept to computer designers ever since Slade and McMahon's 1957 paper[1] described a "catalog" memory. Associative memories offered relief from the continuing problem presented by the typical coordinate-addressed memory which requires that an "address" be obtained or calculated before data stored at that address may be retrieved. The associative memory could acquire in a single memory access any data from memory without pre-knowledge of its location. Ordered files and sorting operations could be eliminated. Unfortunately, early associative memories were expensive, hence none found their way as the "main frame" memory into any commercial computer design.

The organization of an associative memory (AM) requires that each n-bit physical word of the memory be connected to a dedicated processing element (PE) which performs the compare function between a bit read non-destructively from the word and a corresponding input bit from a query word. The PE's for all words are driven by a central controller, thus a single query bit is simultaneously compared with the corresponding stored bit in every word of the AM. With the ability to simultaneously write back the state of each PE into a specified bit position of each word it became possible to perform bit-serial arithmetic between fields of bits within each physical memory word. An array of associative memory words could then be viewed as an array of simple computers—an associative array processor—with all the simple computers in the array simultaneously executing the same instruction obtained from a common control unit as is done in the more complex ILLIAC-IV design.

An alternative AP design provides a PE at each bit of each physical memory word. This design, though complex in terms of logic and interconnection requirements, permits a simultaneous compare of all bits in a query word with all bits of the memory word rather than the serial-by-bit operation described earlier.

Due to the early high cost of semi-conductor memory and logic elements none of the many associative processor designs described in the literature were attractive enough to warrant development. However, it has now become commercially feasible to construct a computing system embodying "main frame" memory content addressability coupled with array arithmetic capability operating under a more or less conventional stored program control system.

Several proprietary versions of the associative processor (AP) are being developed. The first working engineering model[2] known to the author, built for USAF by Goodyear Aerospace Corporation, was demonstrated during a Tri-Service contract review in June, 1969 at Akron, Ohio. The same machine, modified to include a larger instruction memory, was loaned[3] by USAF in 1971 to the FAA for conflict detection tests in a live air traffic control terminal environment at Knoxville, Tennessee operating in a multi-computer configuration with a Univac 1230 conventional computer. The original test objectives were achieved by December, 1971 and additional experiments involving terrain avoidance processing were completed successfully in June, 1972.

The lessons learned in programming and testing the USAF AP model resulted in a new design called STARAN S which was committed to production in 1971. This first commercial AP was publicly introduced in a series of live demonstrations in May, 1972 at the TRANSPO exhibit in Washington, D.C. and in June, 1972 at Boston, Mass.

This paper describes STARAN S and its program-

ming language, provides examples of its applications, and discusses measures of AP cost-effectiveness.

STARAN* DESCRIPTION

A configuration diagram of STARAN S is shown in Figure 1. Studies have shown that initial uses of AP's would be weighted toward real-time applications involving interface with a wide variety of sensors, conventional computers, signal processors, interactive displays, and mass storage devices. To accommodate all such interfaces the STARAN system was divided into a standardized main frame design and a custom interface unit. A variety of I/O options implemented in the custom interface unit include conventional direct memory access (DMA), buffered I/O (BIO) channels, external function channels (EXF) and a unique interface called parallel I/O (PIO).

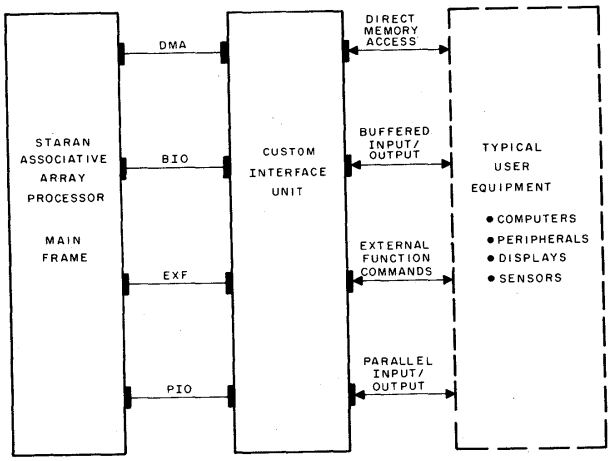


Figure 1—STARAN system configuration

A top-cut diagram of the STARAN main frame is shown in Figure 2. It consists of a conventionally addressed control memory for program storage and data buffering, a control logic unit for sequencing and decoding instructions from control memory and from one to thirty-two modular AP arrays.

A typical AP array is also shown in Figure 2. This key element of the STARAN S computer system is the "main frame" memory which provides content addressability and parallel processing capabilities. Each array consists of 65,536 bits organized as a multi-dimensional access memory matrix of 256 words

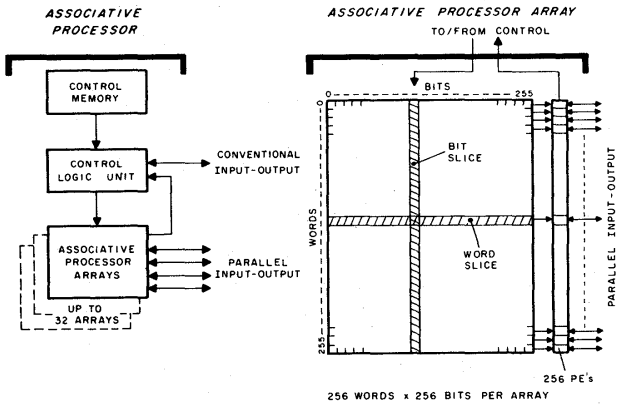


Figure 2—Associative processor diagrams

by 256 bits with parallel access to up to 256 bits at a time in either the word or bit direction. In addition to the storage elements, each array contains 256 bit-serial PE's often referred to in associative memory literature as the response store. The unique PIO capability is provided by the response store, where every PE has an independent external device I/O path. Control signals generated by the control logic unit are fed to the processing elements in parallel and all processing elements execute the instruction simultaneously. As additional arrays are added to the system these are also connected in parallel to the control logic unit, thus application programs need not be modified as the capacity of the system increases.

Major elements of the STARAN block diagram shown in Figure 3 are described below:

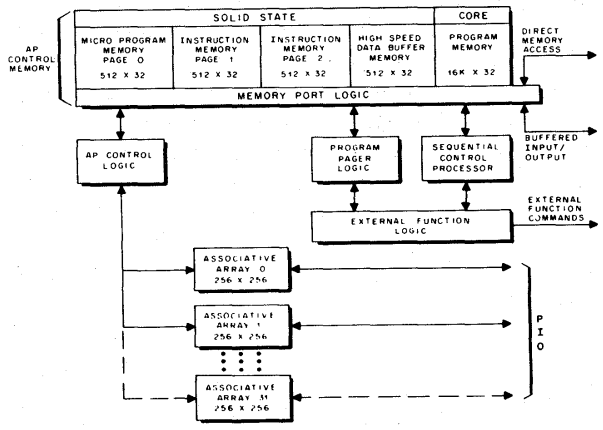


Figure 3—STARAN basic block diagram

* T. M. Goodyear Aerospace Corporation, Akron, Ohio

AP control memory

The conventionally addressed and indexed AP control memory is used to store assembled AP application programs. It is also used for data storage and to act as a buffer between AP control and other elements of STARAN S. The AP control memory and associative array cycles are overlapped.

Control memory is divided into several memory blocks. Three fast "page" memories contain the current AP program segments; the slower core memory contains the remainder of the AP program. A program pager transfers program segments from the slow to the fast memory blocks. Control memory words contain 32 bits of either data or instructions.

The "page" memories use volatile, bipolar, semiconductor elements. A page contains 512 words but can be doubled to 1024 words each on an optional basis. Page 0 may contain a library of microprograms such as arithmetic subroutines. Pages 1 and 2 are used in ping-pong fashion, with AP control executing instructions out of one page while the other is being loaded by the program pager. This permits use of the page memories for selected segments of the program or for the entire program if fast execution is required.

The high-speed data buffer (HSDB), like the page memories, uses volatile, bipolar, semi-conductor elements. It contains 512 words but also can be doubled to 1024 words. All buses can access the HSDB to store data or instruction items that need to be accessed quickly by the different STARAN elements.

The bulk core memory uses nonvolatile core storage. It contains 16,384 words and is optionally expandable to 32,768 words. It is used for storing complete AP application programs. Since the bulk core memory is accessible to all buses it is useful as a buffer for data items that do not require the high-speed of the HSDB.

A block of up to 30,720 AP control memory addresses is reserved for the direct memory access (DMA) channel to external memory. All buses can access the DMA block, thus it is possible to operate the AP solely from programs stored on external memory as, for example, the main frame memory of a conventional computer.

AP control logic

Executing instructions from control memory, AP control logic directly manipulates data within the associative arrays and is the data communication path between control memory and the arrays.

Program pager logic

The program pager loads the fast page memories from the slow core memory. While the AP control is executing a program segment out of one page, the pager can be loading the other page with a future program segment.

External function logic

External function (EXF) logic enables the AP control, sequential control, or an external device to control the STARAN S operation. By issuing external function codes to EXF a STARAN S element can interrogate and control the status of the other elements.

Sequential control processor

The sequential control (SC) portion of STARAN S consists of a sequential processor having an 8K 16-bit memory, a keyboard-printer, a perforated tape reader/punch unit, and logic capability to interface the sequential processor with other STARAN S elements. SC is used for system software programs such as assembler, operating system, diagnostic programs, debugging, and housekeeping routines. SC peripherals which may be useful programming aids are available as options.

Input/output options

A custom interface unit (not shown in Figure 3) can provide any required combination of DMA, BIO, EXF, or PIO channels. A DMA channel to a conventional computer, for example, would permit rapid interchange of data between the systems in the common memory bank. The unique parallel I/O (PIO) channel, with a width of up to 256 bits per array, provides an extreme width channel up to 8192 bits wide at transfer rates in the sub-microsecond region. For example, a four-array STARAN S can input or output 1024-bit word or bit slices at an average slice rate exceeding 3 megacycles/sec providing an I/O bandwidth many times wider than that of a conventional computer. PIO provides a unique capability for large data base processing when used with wide bandwidth mass storage devices.

A photograph of a six array (model S-1500) STARAN is shown in Figure 4.

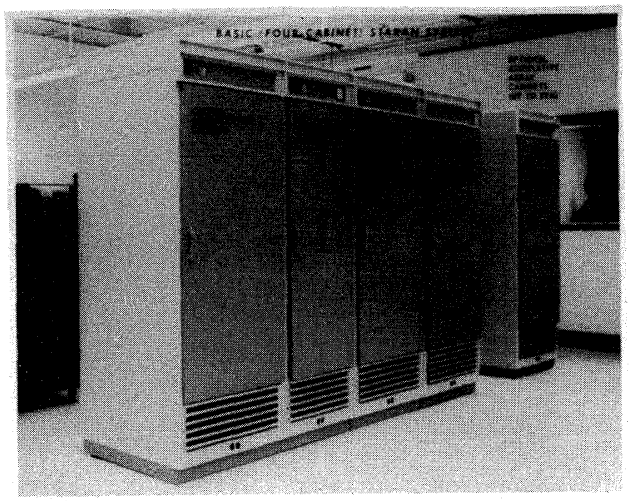


Figure 4—STARAN S-1500

ASSOCIATIVE PROCESSOR SOFTWARE

The STARAN software system consists of a symbolic assembler called APPLE (for **A**ssociative **P**rocessor **P**rogramming **L**anguag**E**), and a set of supervisor, utility, debug, diagnostic, and subroutine library program packages. An associative compiler has not yet been developed for STARAN. Early applications of STARAN must therefore be accomplished by assembly language programmers. Programmers find APPLE a convenient language to use, however, and write significantly fewer instructions to program a suitable application on STARAN than would have to be written for a conventional machine since APPLE's command structure reflects the content addressability and processing characteristics of the associative arrays the language controls. For example, although the programmer must explicitly define his record formats via field definition statements, he usually need not be concerned with physical record location in the arrays. Also, he need not order data tables by key, since any desired datum may be located in one parallel search operation. A third example of APPLE convenience is the elimination of the conventional programming loop which requires advancing a list pointer, examination of an exit criterion, and making a decision for each pass over different data sets. The APPLE array instruction processes all pertinent data sets simultaneously and does not require initialization of an index register with the count of data sets to be processed.

Internally, all software packages with the exception of array diagnostics and the subroutine library operate on the SC. In the minimum STARAN configuration the software packages are furnished on paper tape for input via the SC tape reader. Where STARAN is installed with interface to a conventional computer system in a multicomputer configuration, APPLE and supporting software can be input to STARAN using the existing peripherals of the conventional computer.

The usual load, store, test, branch, and control instructions required for sequential execution of an application program are present in APPLE. Where APPLE departs most from conventional assemblers is in the search and arithmetic array instructions. A representative set of fixed point standard instructions is shown in Table I with the approximate timing formulas. Hardware floating point is available on special order.

Associative search and arithmetic instructions are of two types, "argument register" and "field". In the first an operand (32 bits max) stored in the argument register of AP control is used as the search or arithmetic argument against a specified field in all array words simultaneously. Instructions of the field type perform similar operations but between specified fields within each array word.

Instruction execution times are dependent upon n, the number of bits in the operands (fields) involved in the instruction executions, but are not functions of the number of operands being processed, which relationship is exactly the opposite of that existing in the conventional computer. This characteristic dependence of execution time on operand or field length is a consequence of the word-parallel bit-serial design of the associative arrays discussed earlier.

From the programmer's point of view, Table I has interesting connotations; some of which are:

1. in real time applications the programmer can easily time out his initial flow diagram since programming loops in the conventional sense are eliminated. This single consequence of associative processing can save much of the reprogramming effort invariably found necessary during the testing phase of conventional attacks on real-time problems;
2. he can conserve on execution time (and array memory space) by defining fields to use only as many bits as are required by the application; and
3. he has no need for overhead-generating techniques such as indexed file constructions, linked lists, or sort and merge operations usually needed in a conventional computer. This capa-

TABLE I—Typical APPLE Associative Fixed Point Instructions

MNEMONIC	INSTRUCTION	APPROX. EXECUTION TIME (μs)**			MIPS* PER ARRAY FOR n=32
		FORMULA	n = 16	n=32	
	ARGUMENT REGISTER INSTRUCTIONS				
EQC	EXACT MATCH COMPARAND	0.6+0.15n	3.0	5.4	47
GTC	GREATER THAN COMPARAND	0.7+0.15n	3.1	5.5	47
LTC	LESS THAN COMPARAND	0.7+0.15n	3.1	5.5	47
ADC	ADD AR TO FIELD	2.8+0.85n	16	30	8.5
	FIELD INSTRUCTIONS				
EQF	EXACT MATCH FIELDS	0.6+0.43n	7.4	14	18
GTF	GREATER THAN FIELDS	2.3+0.43n	9.1	16	16
LTF	LESS THAN FIELDS	2.3+0.43n	9.1	16	16
MAXF	MAX FIELDS	0.6+0.68n	11	23	11
MINF	MIN FIELDS	0.6+0.68n	11	23	11
ADF	ADD FIELD TO FIELD	2.8+0.85n	16	30	8.5
MPF	MULTIPLY FIELD BY FIELD	5.8+2.9m+ 0.85mn+0.4	277	980	0.26

* Max execution rate of specified instructions for single array with all 256 PE's active.

** n or m equal number of bits in operand

bility results in a significant reduction both in the number of instructions which must be written and executed and the amount of memory required.

ARRAY STORAGE ALLOCATION

The concept of a file of related records as used in associative processing requires some discussion. In conventional approaches to file generation one thinks of the distinction between a logical file and a corresponding physical file; that is, a logical collection of records, usually ordered by some key, is placed as a block of contiguous addresses in a physical file. The conventional operating system keeps track of the beginning address and the block length for the file whether stored in core or on external stores. Thus in most cases logically different files are stored in physically separate areas of store.

The associative approach differs from the conventional approach in several ways: the records within the logical file need not and usually are not ordered by any key; records within a logical file usually are not stored in contiguous locations in an area of the array or on external devices; and the operating system generally is not required to keep track of individual file beginning addresses and block lengths.

In STARAN, records belonging to different logical files may be physically intermixed in the array as well as being logically unordered. Within each record format, in addition to defining the item fields, the programmer defines a set of control tag fields. How these tags are used is described below.

When new records are added to a logical file the update program writes the new, properly formatted record into the first available empty array location. Since empty array locations usually are not contiguously located within the array, records belonging to a specific file are scattered throughout the array in random locations. This characteristic is illustrated in the array map example of Figure 5.

Empty array memory locations are identified by executing an EQC on a one-bit activity tag field using an "O" as the search criteria. The execution time for this search (see Table I) is less than one microsecond at the end of which time all processing elements for physical memory words containing a 0 in the activity field will be in the "ON" state. At the conclusion of the search a hardware pointer automatically points to

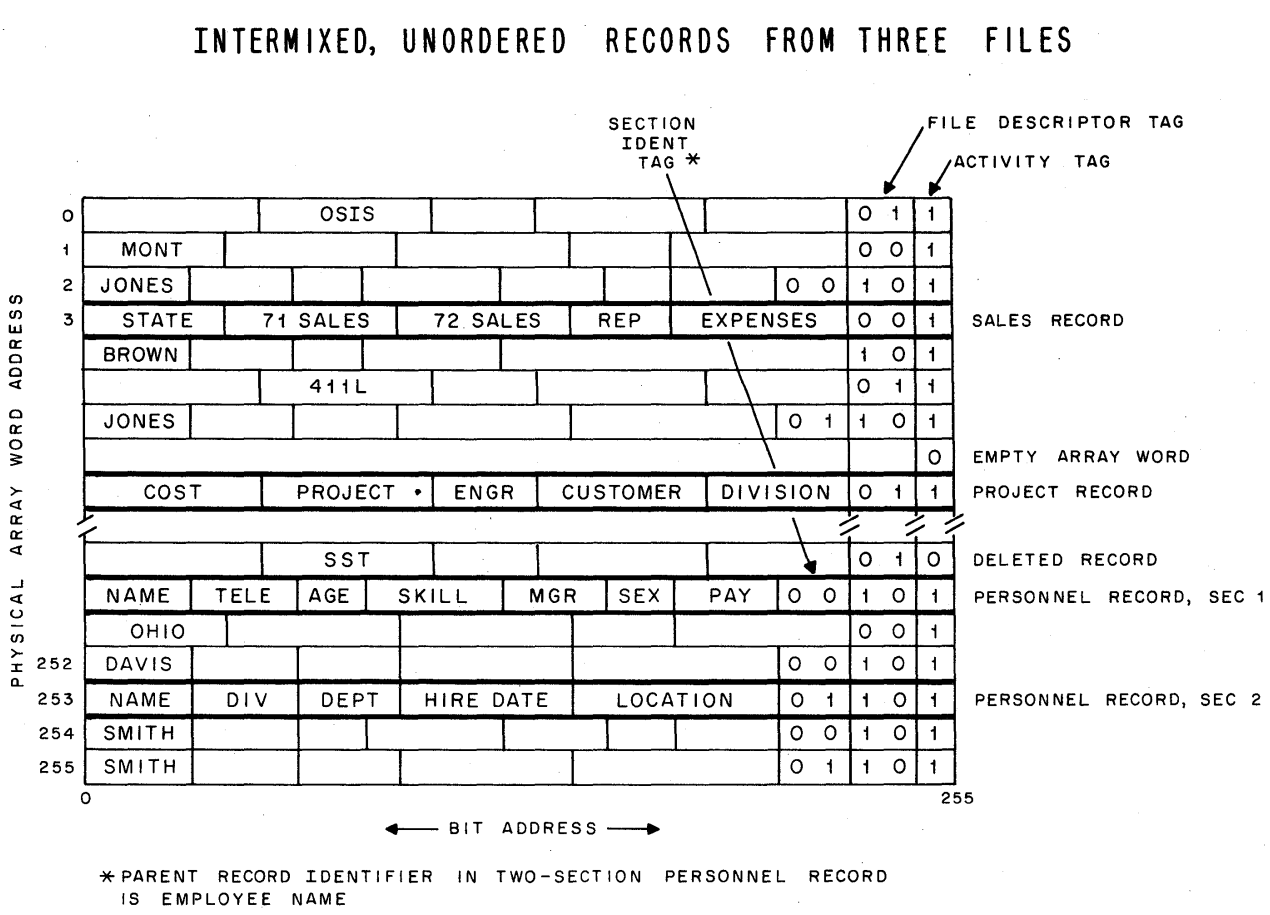


Figure 5—Associative array map example

the PE having the lowest physical address in the array (or arrays). The new record, with its activity field set to a "1," is written into this first empty location. The hardware pointer then moves to the next available empty memory location for writing another record if a batch of new entries must be loaded. If no empty locations are found the program will exit to whatever routine the programmer has chosen for handling this type of error—for example, if appropriate to a specific application, the program may select an age test of all records in a particular file, purging the oldest to make room for the newest. A record once located may be deleted from a file by merely setting the activity bit to an "O."

When a specific file is to be processed in some manner, the scattered locations containing the file's records are activated by performing EQC's on both the activity field and an n-bit "file descriptor" tag field. If, as in the example of Figure 5, the file descriptor field is two bits long, the entire selected file will be ready for processing in less than 2 microseconds (<1 μs for the activity bit search, <1 μs for the file descriptor field search).

Where record lengths are greater than the 256-bit length of the associative array word, several noncontiguous associative array words may be used to store the single record in sections, one section per array word. The format for each record section must contain the same activity and file descriptor fields as are used in all record formats, and in addition it must contain a parent record identifier and an n-bit "section identifier" tag field. The scattered locations containing the desired section of all records in the specific file may be activated by performing EQC's on the activity, file descriptor, and section identifier fields. All three searches can be completed in approximately 2 or 3 microseconds.

These two or three tag search operations in the AP

permit random placement of records in the physical file and eliminate the bookeeping associated with file structuring and control required in conventional systems. The same approach is used for files which exceed the capacity of the associative arrays—the records of such files are stored in a similar manner on external mass storage devices and are paged into the arrays as required.

The strategy used to allocate array storage space can have a significant effect on program execution time. An example is shown in Figure 6 where the products of three operand pairs are required. In A, the operands are stored in a single array word. For 20-bit fixed point operands the three MPF instructions would execute in a total of 1175 microseconds. All similar data sets stored in other array words would be processed during the same instruction execution. However, an alternative storage scheme (B) which utilizes three PE's per data set requires only one MPF execution to produce the three products in 392 microseconds. If one thousand data sets were involved in each case the average multiply times per product would be 392 and 131 nanoseconds, respectively, but at the expense, in B, of using 3000 processing elements. Unused bits in B may be assigned to other functions.

A last example of how array storage allocation can affect program execution time is shown in Figure 7 where the columns represent fields. Here the sum e_1, of 16 numbers is required. If the 16 numbers are directly or as a result of a previous computation stored in the same field of 16 physically contiguous array words, the near-neighbor relationships between the processing elements can be used to reduce the number of ADF executions to four. All similar 16 number sets would be processed at the same time.

STARAN APPLICATIONS

While many papers have appeared (see Minker[4] for a comprehensive bibliography) which discuss the application of AM's and AP's in information retrieval,

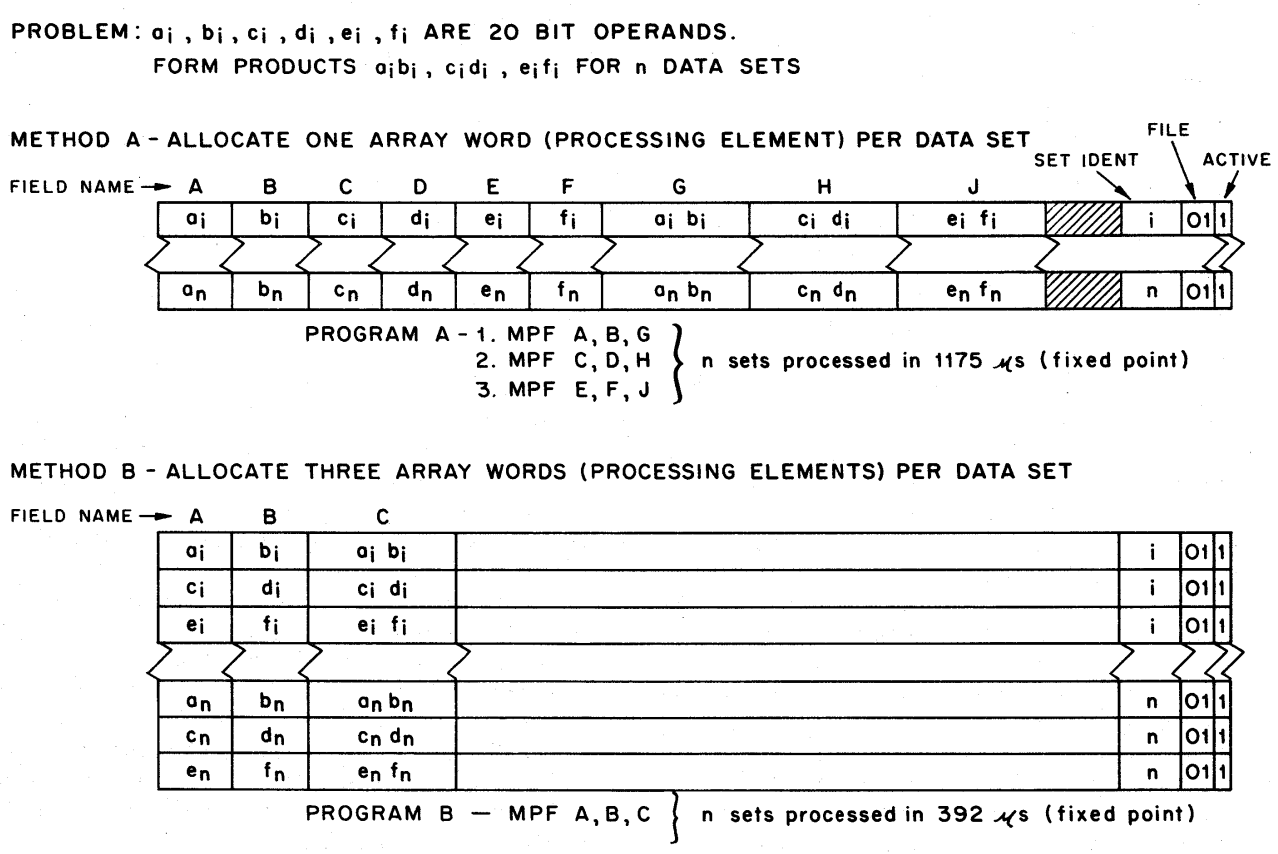


Figure 6—Effect of array memory allocation on execution time

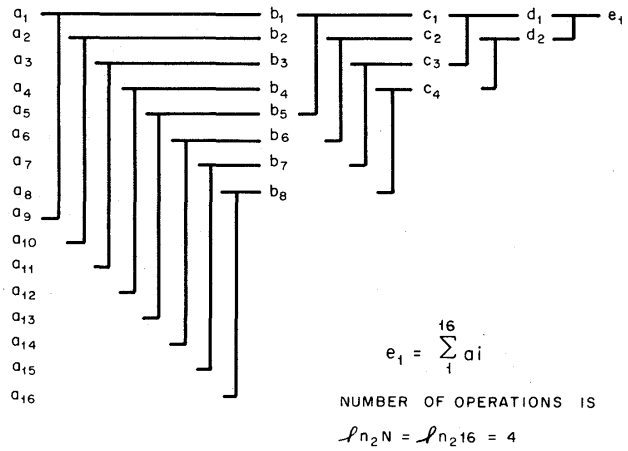


Figure 7—Tree-sum example

text editing, matrix computations, management information systems and sensor data processing systems, there are none yet published which describe actual results with operating AP equipment in any application. (But see Stillman: for a recent AM application result.)

Recent actual applications of the AP have been in real time sensor related surveillance and control systems. These initial applications share several common characteristics:

1. a highly active data base;
2. operations upon the data base involve multiple key searches in complex combinations of equal, greater, between-limits, etc., operations;
3. identical processing algorithms may be performed on sets of records which satisfy a complex search criterion;
4. one or more streams of input data must be processed in real time; and
5. there is a requirement for real time data output in accordance with individual selection criteria for multiple output devices.

A portion of the processing inherent in these applications is parallel-oriented and well suited to the array processing capability of the AP. On the other hand these same applications also involve a significant amount of sequentially-oriented computation which would be inefficient to perform upon any array processor, a simple example being coordinate conversion of serially occurring sensor reports.

Air traffic control

An example of an actual AP application in an air traffic control environment is shown in Figure 8. In this application a two array (512 processing elements) STARAN S-500 model was interfaced via leased telephone lines with the output of the FAA ARSR long range radar at Suitland, Maryland. Digitized radar and beacon reports for all air traffic within a 55 mile radius of Philadelphia were transmitted to STARAN in real time. An FAA air traffic controller's display of the type used in the new ARTS-III terminal ATC system and a Metrolab Digitalk-400 digital voice generator were interfaced with STARAN to provide real-time data output. The controller's keyboard was used to enter commands, call up various control programs and select display options.

Although a conventional computer is not shown explicitly in Figure 8 the sequentially oriented portions of the overall data processing load were programmed for and executed in the STARAN sequential controller as shown in Figure 9. Sequential and associative programs and instruction counts for STARAN are shown in Table II. In a larger system involving multiple sensors and displays, and more ATC functions such as metering and spacing, flight plan processing, and digital communications, the sequential and parallel workloads would increase to the point where a separate conventional computer system interfaced with the AP would be required.

The STARAN system was sized to process 400 tracks. Since the instantaneous airborne count in the 55 mile radius of Philadelphia was not expected to exceed 144 aircraft, a simulation program was developed to simultaneously generate 256 simulated

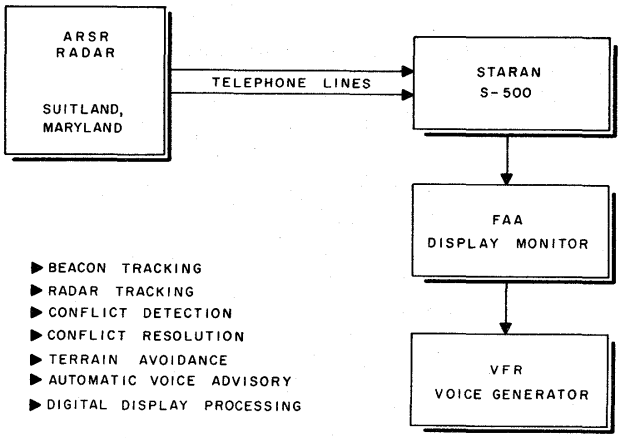


Figure 8—Air traffic control application

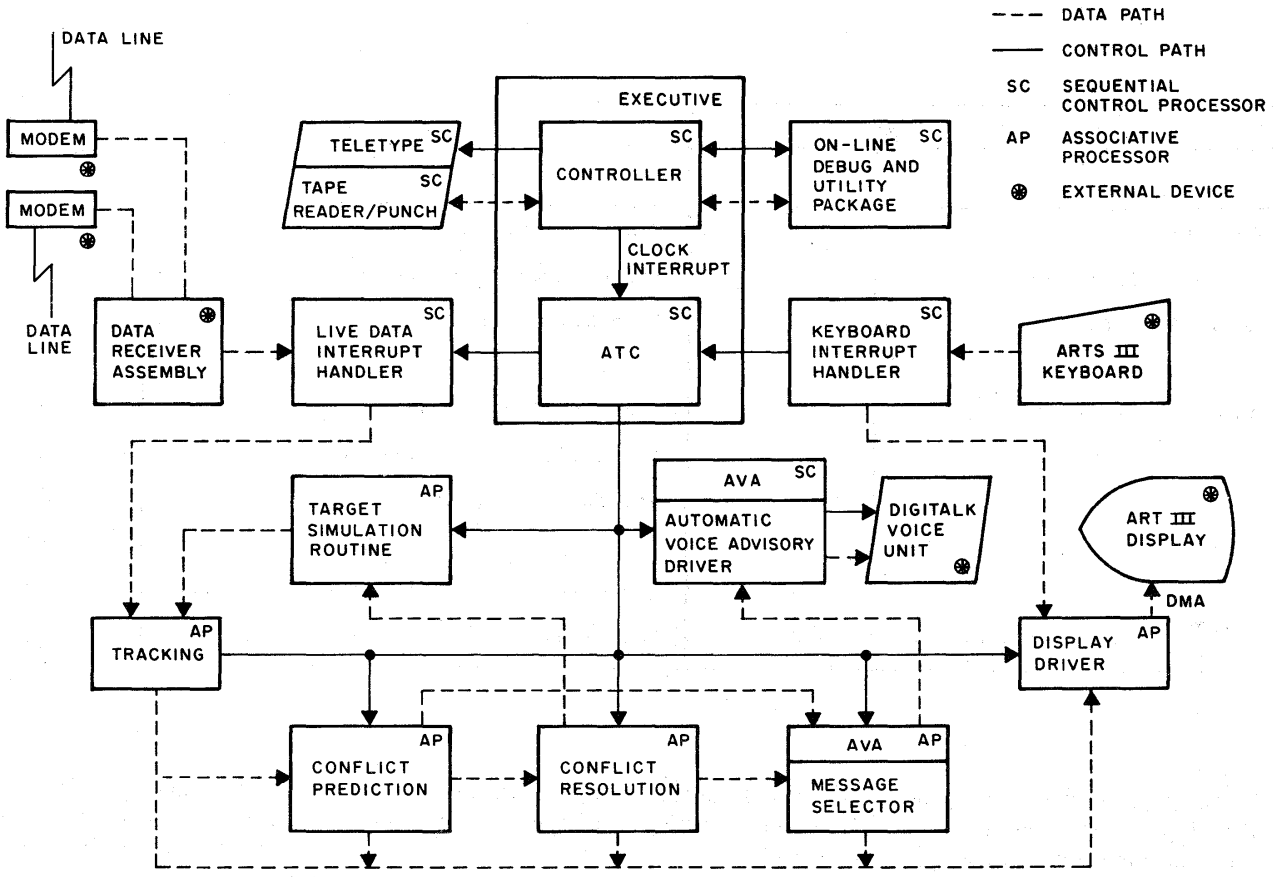


Figure 9—ATC program organization

aircraft tracks. Display options permitted display of mixed live and simulated aircraft. The 400 aircraft capacity is representative of the density expected as North-South traffic loads increase through the late '70s. Conflict prediction and resolution programs based upon computed track data were demonstrated and used to display conflict warning options. Automatic voice services were provided for operator-designated aircraft, thus simulating warning advisories for VFR pilots requesting the service. The voice messages, which in an operational system would be automatically radioed to the pilot, were generated by the Metrolab unit from digital formats produced by the associative processor and broadcast in the demonstration area via a public address system. A typical message would be read out in voice as, "ABLE BAKER CHARLIE, FAST TRAFFIC SEVEN O'CLOCK, 4 MILES, ALTITUDE 123 HUNDRED, NORTHEAST BOUND".

Top level flow charts for four of the associative programs used in the demonstration are shown in Figures 10, 11, 12, and 13. A detailed report is in preparation describing all of the ATC programs used in this demonstration, but some comments on the four flow charts shown may be of interest.

Live target tracking (Figure 10) is performed in two dimensions (mode C altitude data was not available) using both radar and beacon target reports to track all aircraft. Incoming reports are correlated against the entire track file using five correlation box sizes, three of which vary in size with range. Any incoming report which does not correlate with an existing track is used to automatically initiate a new tentative track. An aircraft track must correlate on two successive scans and have a velocity exceeding 21 knots to qualify as an established track and must correlate on three successive scans to achieve a track firmness level high enough to be displayed to a controller as a live target.

TABLE II—STARAN Air Traffic Control Programs

SEQUENTIAL PROGRAMS		ASSOCIATIVE PROGRAMS	
NAME	INSTR COUNT		INSTR COUNT
Executive	1600	Tracking System	881
Keyboard Interrupt		Track Simulation System	415
Real Time Interrupt		Turn Detection	88
Live Data Input		Conflict Prediction	488
Automatic Voice Output		Conflict Resolution	296
		Automatic Voice Advisory	709
		Display Processing	1140
		Total	4017
		Field Definition Statements Included	514
Net Operating Instructions	1600	Net Operating Instructions	3493

There are provisions for 15 levels of track firmness including 7 "coast" levels. If a report correlates with more than one track, special processing (second pass resolve) resolves the ambiguity. Correlated new reports in all tracks are used for position and velocity smoothing once per scan via an alpha-beta tracking filter where for each track one of nine sets of alpha-beta values is selected as a function of track history and the correlation box size required for the latest report correlation. If both beacon and radar reports correlate with a track, the radar report is used for position updating. Smoothed velocity and position values are used to predict the position of the aircraft for the next scan of the radar and for the look-ahead period involved in conflict prediction.

Track simulation processing (Figure 11) produces 256 tracks in three dimensions with up to four programmable legs for each track. Each leg can be of 0 to 5 minute duration and have a turn rate, acceleration, or altitude rate change. A leg change can be forced by the conflict resolution program to simulate pilot response to a ground controller's collision avoidance maneuver command. Targets may have velocities between 0-600 knots, altitudes between 100-52,000 feet, and altitude rates between 0-3000 feet per minute.

The conflict prediction program sequentially selects up to 100 operator-designated "controlled" or "AVA" aircraft, called reference tracks in Figure 12, and compares the future position of each during the look-ahead period with the future positions of all live and simulated aircraft and also to the static position of all terrain obstacles. Any detected conflicts cause conflict tags in the track word format to be set, making the tracks available for conflict display processing. A turn detection program not shown opens up the heading uncertainty for turning tracks.

Display processing (Figure 13) is a complex associative program which provides a variety of manage-by-exception display options and automatically moves operator-assigned alpha numeric identification display data blocks associated with displayed aircraft so as to prevent overlap of data blocks for aircraft in close proximity to one another on the display screen. Sector control, hand off, and quick-look processing is provided.

All programs listed in Table II were successfully demonstrated at three different locations in three successive weeks, using live radar data from the Suitland radar at each location. The associative programs were operated directly out of the bulk core and page 0 portions of control memory since there was no requirement, in view of the low 400 aircraft density

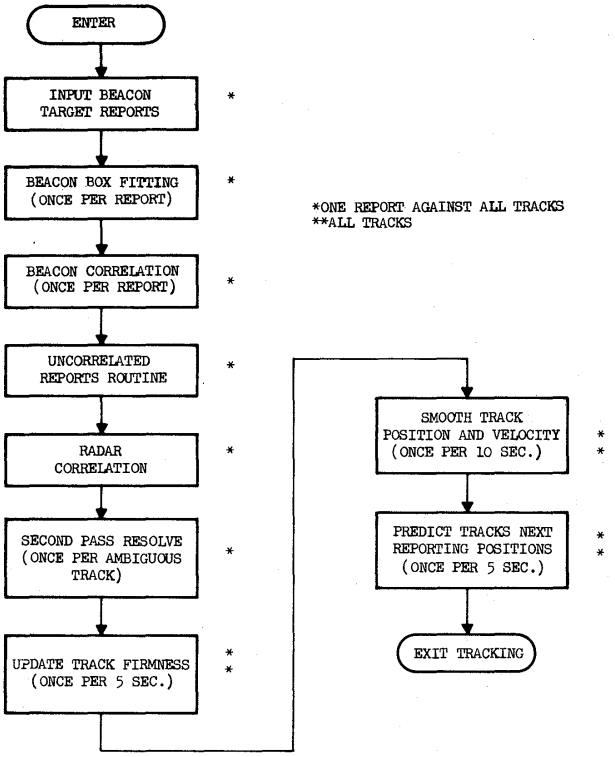


Figure 10—Live target tracking

involved, for the higher speed instruction accesses available from the page memories. At intervals during the demonstration all programs were demonstrated at a speed-up of 20 times real time with the exception of the live data and AVA programs which, being real-time, cannot be speeded up. Timing data for the individual program segments will be available in the final report. The entire program executed in less than 200 milliseconds per 2 second radar sector scan or in less than 10 percent of real time. All programming effort was completed in 4½ months with approximately 3 man-years of effort. This was the first and as of this writing the only actual demonstration of a production associative processor in a live signal environment known to the author. It was completed in June, 1972. Other actual applications currently in the programming process at Goodyear involve sonar, electronic warfare and large scale data management systems. These will be reported as results are achieved.

COST EFFECTIVENESS

Associative processor cost effectiveness can be expressed in elementary terms as shown in Figure 14 where performance is shown in terms of millions of instructions per second for the ADF and EQC instructions using two different operand lengths, and cost effectiveness is measured in terms of instructions per second per hardware dollar. This form of presentation was taken from Bell.[6]

Another cost effectiveness measure is to compare projected hardware and software costs of an associative configuration and an all-conventional design for the same new system requirements, where the associative configuration may include a conventional computer. Only a few attempts at this approach have been made to date and none have been confirmed through experience. One classified example, using a customer defined cost effectiveness formula, yielded a total system cost effectiveness ratio of 1.6 in favor of the associative configuration.

Of the two methods, the first is least useful because there is no way of estimating from these data how much of the associative computing capability can be used in an actual application. The second method is

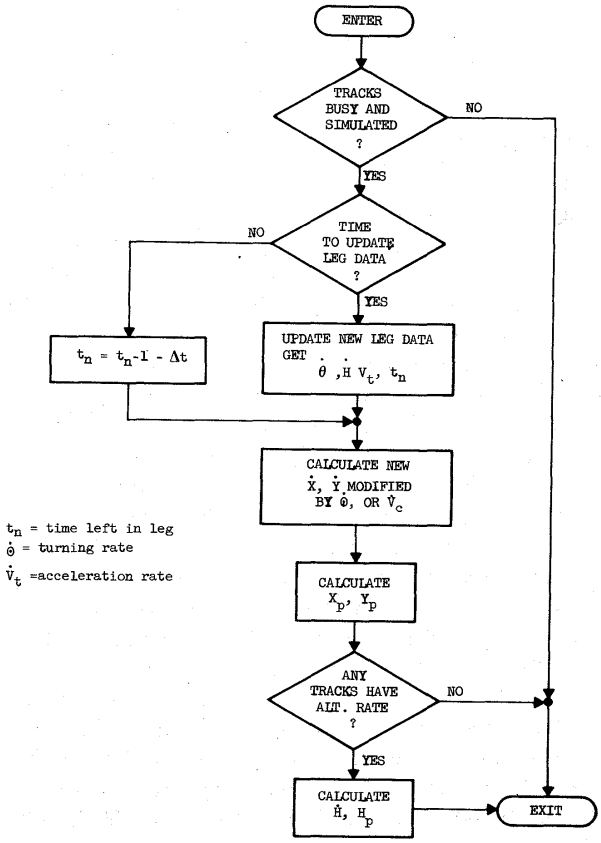


Figure 11—Tracking simulation

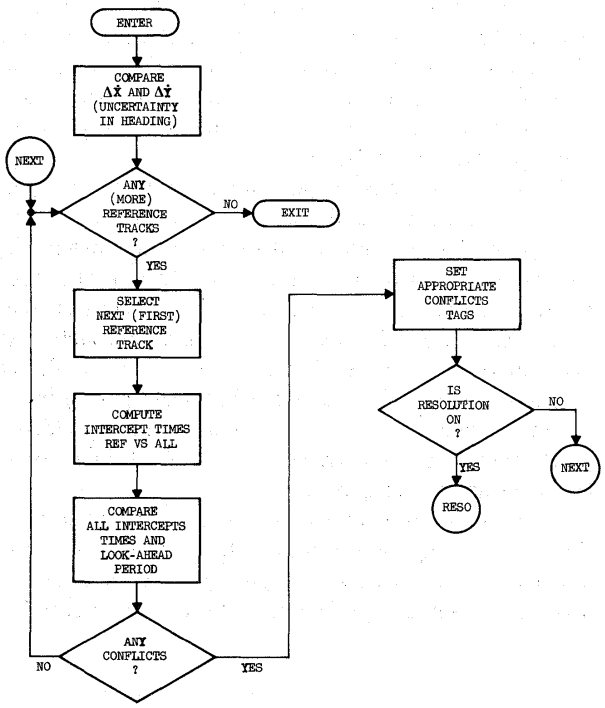


Figure 12—Conflict prediction

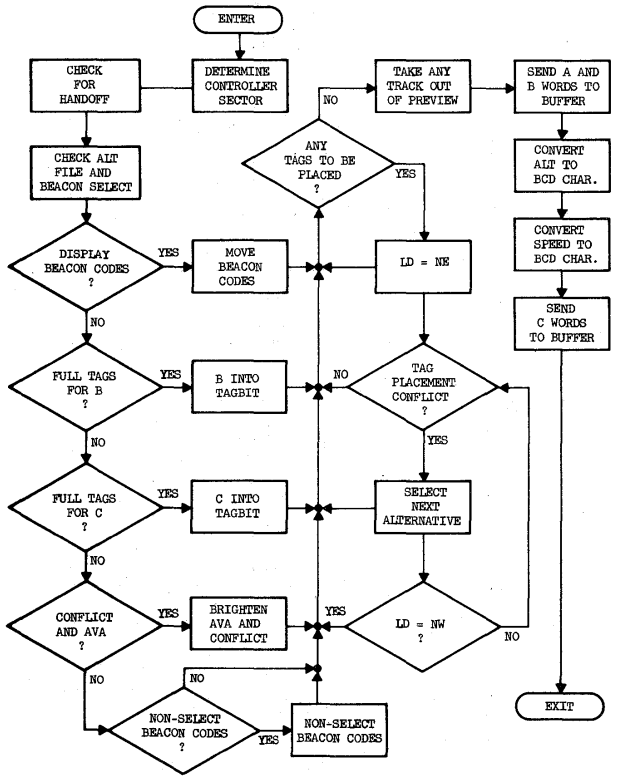
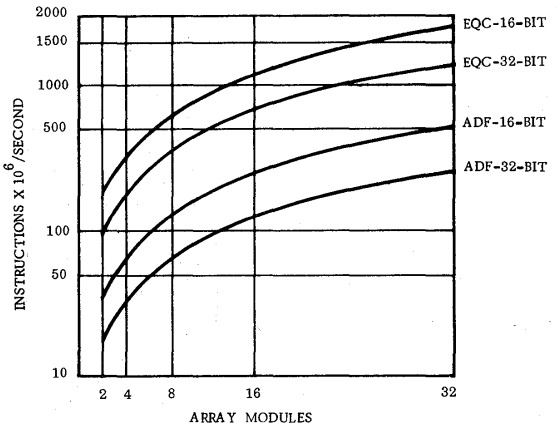


Figure 13—Display processing

more meaningful but is exceedingly expensive to use since it implies a significant engineering effort to derive processing algorithms, system flow charts, instruction counts, and timing estimates for both the conventional and the associative approach. The weakest element in this approach lies in the conventional approach software estimate which historically has been subject to overruns of major dimensions.



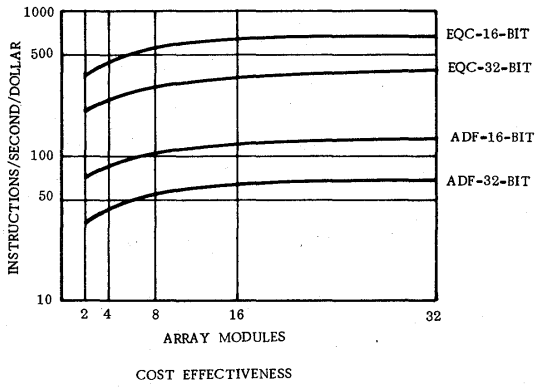


Figure 14—Array performance and cost effectiveness

A third method is to compare functional performance, hardware and software cost, growth capability and growth related costs, reliability, service and other pertinent aspects of two working examples of competing approaches to the same class of system application. Although it is a reliable method, it is not available at this time since no operational system of any kind has been implemented with an associative processor. The closest approach to it is the ATC demonstration described above but there is no similar conventional example to be found anywhere which includes the urgently needed large scale conflict detection process-

ing included in the STARAN demonstration. On the other hand, an experienced ATC data processing system designer can appreciate the rapid solution time, small instruction count and low programming cost achieved with the STARAN for the troublesome high density tracking and display processing functions, but others not so well acquainted with ATC data processing problems may not find these data meaningful. This method also includes the benchmark test which is coming into regular use by the federal government in competitive large scale procurements of standard commercial equipment. Here again, however, due to the associative processor's recent arrival on the scene, no comparative performance data are yet available.

A fourth method, least useful in resolving the equipment selection and system design problems involved in a specific near term application, is based upon theoretical machine design considerations such as gate count ratios, logic to memory ratios and hardware efficiency or duty cycle ratios for conceptual machines which have not been reduced to practice during the typical seven year development cycle for new computer architecture.

Thus, until near term potentially cost effective associative processor applications are accomplished in operational environments, comparative cost effectiveness analyses of proposed associative versus conventional solutions will continue to be suspect. The next 12 to 18 months should produce a substantial improvement in the availability of reliable cost and performance data for associative processor applications.

SUMMARY

Although several manufacturers are developing associative processor equipment, the first version to be produced in a production configuration was introduced in May of 1972 by Goodyear Aerospace Corporation following FAA on-site tests in 1971 at Knoxville, Tennessee of a USAF-owned engineering model built and demonstrated by Goodyear in 1969.

The processor provides full content addressability and array arithmetic capability within "main frame" memory coupled with a unique capability for wide bandwidth (over 3000 megabits/sec for a 4-array STARAN) input-output data transfers to mass data stores. The associative programming language, APPLE, provides a flexible and convenient assembler for programming array arithmetic and search algorithms without the complex and costly indexing, nested loop and data manipulation constructions required in conventional computer programming.

The associative processor may be viewed as a software-programmable super-peripheral, or special purpose subsidiary processor, for attachment to any general purpose conventional computer system via standard channel attachment. In this role the super-peripheral is assigned parallel oriented problem segments and data bases which would otherwise, through excess operating system software overhead, tend to choke the conventional machine.

Although first applications of the associative processor are of the real time, dedicated, command and control type, the extension to large scale data base management, on-line management information systems with immediate response to complex multiple-key queries, and large scale matrix computations await only user decision and ingenuity to accomplish now that production hardware and software has become available at the 370/145 price level.

The cost effectiveness of associative processing has yet to be proven in operational systems, but test results from initial users should accumulate rapidly now that associative processing is no longer only an interesting concept in the literature.

REFERENCES

1 A E SLADE H O McMAHON
The cryotron catalog memory system
Proc 1957 FJCC Vol 10 pp 115-120

2 L C FULMER W C MEILANDER
A modular plated wire associative processor
Proc IEEE Computer Group Conference June 1970

3 J A RUDOLPH L C FULMER
W C MEILANDER
The coming of age of the associative processor
Electronics February 15 1971 pp 91-96

4 J MINKER
A bibliography of associative or content-addressable memory system: 1956-1971
Auerbach Corporation 121 N Broad Street Philadelphia Pa 19107 June 15 1971

5 N J STILLMAN
Associative processing and computer graphics—A feasibility study
USAF Report RADC-TR-72-57 April 1972

6 C G BELL R CHEN S REGE
Effect of technology on near term computer structures
Computer March-April 1972 pp 29-38