AD/A-003 414

EXPERIENCES WITH AN OPERATIONAL ASSOCIATIVE PROCESSOR

D. L. Baldauf

Mitre Corporation

Prepared for:

Electronic Systems Division

November 1974

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

AD/A003 414

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ESD-TR-74-199	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Experiences with an Operational Associative Processor		5. TYPE OF REPORT & PERIOD COVERED
		6. PERFORMING ORG. REPORT NUMBER MTR-2879
7. AUTHOR(s) D. L. Baldauf		8. CONTRACT OR GRANT NUMBER(s) F19628-73-C-0001
9. PERFORMING ORGANIZATION NAME AND ADDRESS The MITRE Corporation Box 208 Bedford, Mass., 01730		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS Project No. 572T
11. CONTROLLING OFFICE NAME AND ADDRESS Deputy for Command and Management Systems Electronic Systems Division, AFSC L. G. Hanscom Field, Bedford, Mass., 01730		12. REPORT DATE November 1974
		13. NUMBER OF PAGES ~~35~~ 37
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

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

Associative Processing
Parallel Processing
Array Processor
STARAN

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

A space object position prediction program was implemented on the STARAN associative array processor (AP) installed at the Rome Air Development Center (RADC), New York. This document outlines the experience gained from this task. A section is devoted to an analysis of the time and effort required to implement the program. Emphasis is given to the program design and array layout phase. Systematic (i.e., independent of the specific program) and application-related capabilities and limitations are discussed. An analysis of the RADCAP system from a user's viewpoint

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

(37)

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

is also presented. The latter part of the paper deals with recommendations for an improved STARAN system (hardware and software) and an improved host computer interface.

ia

REVIEW AND APPROVAL

"This technical report has been reviewed and is approved for publication."

[illegible]

MARVIN E. BROOKING, GS-13
Task/Project Officer

FOR THE COMMANDER

[illegible]

ROBERT W. O'KEEFE, Colonel, USAF
Director, Information Systems Technology
Applications Office
Deputy for Command & Management Systems

ACKNOWLEDGEMENTS

Special thanks are due to Paul Gamelin for his aid in the production of this document. Thanks also go to Robert Fiske, Paul Gregor and Gerald Koehr of the MITRE Corporation for their support. The cooperation of those at the Rome Air Development Center, especially Armand Vito, is gratefully acknowledged. The work described was sponsored by the Air Force Systems Command/Electronic Systems Division/ Information Systems Technology Application Office.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

SECTION I

INTRODUCTION

During FY74 principal efforts of the Associative Processor Task were directed toward implementing the SGP4 space object position prediction equations on the Rome Air Development Center Associative Processor (RADCAP) facility. The RADCAP facility consists of a STARAN(*) S-1000 associative processor interfaced to an HIS-645 Multics system. Results of these efforts are discussed in this report.

The SGP4 (Simplified General Perturbations #4) system is used to predict the position of orbiting space objects. Computations are based on a simplified version of Lane and Cranford's drag theory[1] combined with a simplified version of Kozai's geopotential theory[2] along with second order secular terms from Brouwer's geopotential theory[3]. The program takes a set of mean elements at an epoch time and a drag term and uses them to generate position and velocity at some specified future time. The SGP4 equations are typical of a class of scientific computations with a branchless algorithm. In this type of computation, a series of arithmetic computations are applied to each of many data groups. For the space object tracking problem, several hundred groups of data were processed with each data set containing information pertaining to one space object. Some of the characteristics of the STARAN were found to be systematic, i.e., they were independent of the program being implemented. Others were found to be directly related to the particular type of program being implemented.

(*)STARAN is manufactured and marketed by Goodyear Aerospace Corporation, Akron, Ohio.

Preceding page blank

Systematic and application-related capabilities are pointed out throughout the paper. A section is also devoted to an analysis of the time and effort required to implement the program on the STARAN. These should be useful in guiding any subsequent efforts to implement similar algorithms on the RADC system.

The SGP4 equations are characterized and results of their application are discussed in Section II. Section III traces the proposed solutions, the final solution, and the time needed for all phases of the experiment. Section IV evaluates the RADC STARAN system architecture from a general user's viewpoint and from the results of the SGP4 experiment. Summary and conclusions are presented in Section V. Appendix A presents a description of the STARAN hardware and software available at RADC. Appendix B explains the specially developed PIO routines.

SECTION II

SGP4 PROBLEM DEFINITION

EQUATIONS

The SGP4 equations (Simplified General Perturbations #4) are used by the Space Computation Center (SCC) for space object orbital position prediction. There are 44 equations containing 259 multiplications, 30 divisions, 62 additions, 42 subtractions, 15 square roots, 7 sines, 6 cosines and 1 arctangent. There is, of course no "real" number of arithmetic operations for a set of equations; these are the numbers that were found for the particular way that the equations were programmed in this project. Shortcuts such as Newton's method for polynomials were taken wherever possible. Expressions used in one equation and needed later were saved. This reduced the number of arithmetic operations. Each of the trigonometric functions were solved by computing a truncated series expansion. The arithmetic operations necessary to compute this series were not included in the numbers above. Fourteen pieces of input data are associated with each space object. These data contain such information as the object identifier, epoch time, shape and orientation of the orbit and drag coefficient. The same calculations are performed for each object. There are no branches in the code, and 16 results are produced. These results are used in other programs of the SCC such as ephemeris generation and look angle reports for ground based sensors. These equations are typical of a large class of scientific computations which exhibit the same branchless type algorithm. Thus, many of the items discussed in this paper concerning this implementation are applicable to other problems.

APPROACH TO PROBLEM

It was assumed that several thousand space objects were to be processed. The goal of the implementation effort was to minimize the effective processing time per object. The intelligent use of the parallel capabilities of the STARAN was clearly the method to be used to achieve this. The equations were examined for possible sources of parallelism. Matrix operations are one possible source. DO LOOP constructions are another source of parallelism. Neither of these were found within the equations. It was then decided to make use of the other major source of parallelism - parallelism by number. In this case, many data groups go through the same series of arithmetic computations. For the purposes of the experiment each data group corresponds to one space object. Many objects can be processed simultaneously by using the parallel processing capabilities of the arrays.

PROPOSED SOLUTIONS

Choosing the best algorithm, given the RADCAP configuration, was the next task. Several designs were considered. All designs assumed 32-bit floating point numbers as data.

The first and most obvious approach taken was the use of one word (256 bits) of associative memory per object. Since there are 1024 words of associative memory (4 arrays with 256 words/array) this would allow the processing of 1024 objects in parallel. Several problems were presented with this design. There are 14 inputs and 16 outputs in the SGP4 equations. Since all arrays are being used for computation, I/O through the parallel input/output (PIO) channel cannot be overlapped with computation. Sixteen outputs from each of 1024 objects also requires 16K words of control memory for storage. This is exactly the size of memory and leaves no room for a program. In addition, many temporary variables are produced during the computations and are needed in subsequent calculations. Two hundred fifty six bits (eight 32-bit fields) is not sufficient storage to hold all of these variables. Some must be written out to control memory and subsequently read in. This operationsis very time consuming.

A second design which would eliminate the storage problem was proposed. Instead of using one word of associative memory for each object, two contiguous words are used. This doubles the storage available for temporaries in the array. This also reduces the storage needed in control memory for the 16 outputs from 16K to 8K. However, only 512 objects can now be processed at once. Once again, since all arrays are being used for computation, I/O cannot be overlapped with computing. A few temporaries must still be stored in control memory but not nearly as many as would be necessary with the first design.

The third design utilizes the ability of the AP control and the PIO control to execute simultaneously. The arrays are functionally divided in half. Arrays 0 and 1 are computation arrays, and arrays 2 and 3 are buffer areas. As in the second design, each object uses two words of associative memory; one word is in the computation array, the other is in the corresponding word of the buffer array. The word in the buffer area is used both for storage of temporary variables and for input/output operations. Arrays 2 and 3 are no longer used for computation. Thus, for most of the time they can be assigned to PIO control to do input/output. At the same time, arrays 0 and 1 are computing. In this manner most of the I/O is being overlapped with computation. In the execution of the program, data is first brought into the computation arrays from control memory. Computation begins while the buffer arrays continue to bring in input variables. As more variables are needed for computation, processing is interrupted and the variables are transferred from the buffer arrays. Similarly, when temporary variables are not needed for a time, they are transferred from the computation arrays to the buffer arrays. The third design was judged to be the best of those considered because it:

1. Allowed AP control and PIO control to execute simultaneously.
2. Provided 16 fields of array storage.
3. Reduced control storage needed for variables.

SECTION III

IMPLEMENTATION

The first phase of the implementation effort was devoted to a study of the RADCAP system and the SGP4 equations. This study required about 180 man-hours. This time was spent on investigating the different approaches to the solution as discussed in the previous section. Rough estimates for time and throughput were computed for each of the three designs. By taking the time to look into approaches that were not immediately obvious, an increase in throughput of approximately 50% was realized. Because of this savings it is suggested that this phase be emphasized in similar implementation efforts.

The data area map was next formulated. Since there is no higher level language for STARAN, all memory management must be done by the programmer. This required approximately 150 man-hours and involved both the array area and the control memory area. A decision was made to use 32-bit floating point numbers for all data. This meant that there were only eight fields in each of the arrays. Since a typical arithmetic operation involved three fields, it can be seen that the allocation of storage in the arrays was a major concern. It became necessary to go through the equations operation-by-operation in order to place exactly every piece of information within the arrays. Storage had to be allocated for temporary variables that were produced. Some items had to be moved several times between control memory, buffer arrays and computation arrays. All of these operations had to be carefully planned to make the best use of the computation and I/O capabilities of the system. Care had to be taken that no mistakes were made while transforming the equations into array operations. A missed addition, for example, would upset the carefully planned array layout. A fix might have to be made by transferring a word in the computation arrays to the buffer arrays. This type of time-consuming change also makes it difficult to make additions or changes to a program that has already been programmed for the STARAN. Thus, the code produced in this type of program is rather inflexible to change. Because of the importance of properly utilizing all of the resources of the STARAN, this phase should be important in any similar task.

The coding part of the project was relatively straightforward once the data area mapping was complete. This required about 60 man-hours. The code was produced almost directly from the array map. A second pass was made to try to optimize that code relative to the I/O operations which were taking place. Whenever possible, I/O operations were put together in a group. Appendix B more fully explains this process. Also, wherever possible, input/output commands were placed in front of a series of arithmetic operations. Thus, when PIO control was finished executing, AP control would still be processing the arithmetic commands. Both of these changes allow PIO routines and AP routines to run with little interference with each other.

It was then necessary to break the code into sections. The entire program was too large to assemble on the STARAN. Because of the slow speed of the macro preprocessor, smaller, more manageable pieces of code had to be assembled; otherwise, the time to find syntax errors and reassemble would have been prohibitive. This division of the code caused additional problems. Assembler directions such as EQU statements and define field instructions had to be repeated for each section. Certain important constants had to be made ENTRYs to enable references from separately assembled sections of the code. An overlay had to be generated which resulted in the later addition of some code. The total time for these operations was fifteen hours.

The selected design could not execute properly without the addition of some input/output system software. PIO routines were needed to transfer information between arrays 2 and 3 and control memory while AP control was executing on arrays 0 and 1. In addition, AP control programs had to be developed to call the I/O routines. A parameter passing standard had to be developed. Since the PIO routines could be called in a variety of ways, it was also necessary to develop complex macros. Chaining of commands was also allowed. A detailed description of the PIO routines appears in Appendix B. Approximately 130 man-hours were required to develop these routines.

The debugging time consisted mainly of the time required to debug the parallel input/output routines. Problems occurred with finding infinite loops in the code. It was necessary to halt the STARAN in order to reload the system when loops occurred. Since breakpoints could not be set in PIO control, routines had to be run until completion in order to view results. Also, registers could not be modified to try different changes. Array and register dumps were mainly utilized in obtaining a properly executing program. Since the emphasis of the work was not on obtaining a "correct" answer, time to debug logic errors is not included. Sixty man-hours were devoted to the debugging task.

SECTION IV

CAPABILITIES AND LIMITATIONS

SOFTWARE

The macro pre-processor on the STARAN is a very powerful tool which can greatly simplify any programming task. It contains many useful features some of which are listed below.

- Ability to nest macro calls (including recursively)
- Ability to have a macro call appear as a character string variable
- Local and Global macro variables
- System macro variables for label field, argument field and a SYSNDX type construction
- Arithmetic and character string literals
- Subscripts for indexing into character strings
- Substrings
- Logical and arithmetic operators
- Concatenation
- Conversion between string types
- Ability to limit number of macro branch instructions executed
- Message generation facility

The STARAN assembler is very easy to use. It has the typical assembler fields - label field, command field, operand field and comment field. All lines are free format with the only requirement being that labels must start in the first column. A useful feature is the ability to equivalence variables to registers as well as symbols. Some registers in PIO control have the same names as those in AP control but have different functions. The equivalence feature allows a more mnemonic labeling of these registers. Another feature is the ability to intersperse machine code with assembly code within a program and within an instruction. This is sometimes necessary because the assembler cannot generate all of the possible machine language instructions. In most applications this would not affect the programmer since the assembly language is sufficiently powerful. However, in applications requiring new system software (i.e., PIO routines) machine language must be extensively used.

A useful addition to the assembler would be a DSECT or COMMON type construction that would allow description of data areas which are external to a program. This would be useful when several independently assembled programs or sections of programs refer to a common data area. With the existing assembler, all such variables referenced must be made external variables.

The linker is powerful and easy to use. Overlays are easily generated and used with the aid of simple assembly language statements. Either high or low load locations can be specified for each segment to allow optimal use of control memory.

The debugging facilities provided with the STARAN are generally very good. Especially useful are the breakpoint and print table facilities. However, more breakpoints and the ability to set PIO breakpoints would be helpful. The print table allows pre-selected registers and locations to be dumped upon user command. Every register and memory location - array, AP control, PIO control and sequential processor control can be examined. Changes can be made to all locations and registers except PIO registers. This proves to be a handicap when debugging input/output routines. There is also a single step mode to execute programs one instruction at a time. This is a very helpful feature, but problems can occur. There are loop instructions in STARAN which allow the repeating of groups of instructions up to 256 times. When debugging a program in single step mode, each of these instructions within the loop must be manually stepped through as many times as is indicated for the loop. A mechanism to suspend single stepping within the loop and to resume it outside of the loop would be helpful. Another shortcoming of the present debugging system is the absence of any in-line debugging aids. Items such as snapshot dumps of registers and locations, especially array memory, could be very useful. In addition, conversion of floating point data to a standard output format would aid in debugging programs that use floating point arithmetic. As the system exists now, all debugging must be on-line. Debugging in batch mode is impossible. For remote users, this especially presents problems because of the way that the STARAN - host machine (Honeywell 645) interface is handled. During on-line debugging it is necessary to know the status of the machine at all times. A user essentially asks the machine a question, it responds, and he deductively proceeds from there to ask more questions. The feedback from the computer is very important. In stand-alone debugging, this feedback is present. When remotely accessing the STARAN, only some of the feedback occurs. No immediate response is given when starting the AP, halting the AP, stopping at breakpoints or reading the performance monitor. In order to verify that these instructions have been carried out, debug mode must be left completely. This greatly slows down the debugging effort. On occasional system crashes no notice is given to the user. There is also no provision for a restart of the STARAN from the host computer. This could greatly simplify the procedure used to reload the system after a crash. When doing any work remotely on the STARAN, the user is locked out of any other Multics activity. For example, while waiting for a program to assemble on STARAN a user cannot use a Multics editor to create another program. Also, a user may wait so long for a program to be assembled that he gets disconnected from the STARAN and logged off the Multics system. There also does not exist the capability of running a program on Multics that interacts with a program on STARAN. This kind of interaction could be helpful in a program that is only partially suitable for an AP. Some of the processing could be done under Multics, information could be sent to STARAN and a parallel task initiated. Information to guide processing could be communicated between the two tasks. However, with the present system this is impossible.

Another obvious shortcoming of the software is the lack of a higher level programming language. The burden now is entirely on the programmer to take his problem down to the machine level. A higher level language could greatly simplify his task. Also, as mentioned before, the code produced is rather inflexible to change because of the array allocation.

HARDWARE

The most important recommendation for a modified RADC system is an increase in associative array word size. The storage of temporary variables was the most difficult problem to deal with in the implementation effort. Even the use of buffer arrays for additional temporary storage did not eliminate the necessity of storing temporary variables in control memory. Input/output operations to core were very slow when compared to arithmetic operations. The addition of more storage per array word could eliminate entirely the need to use control memory for temporary storage. It was estimated that doubling the word size of array memory to 512 bits would have this effect. Since two arrays are being used in the implementation as buffer areas, another recommendation would be to remove the logic from several arrays and attach them to the present arrays as input/output buffers. These recommendations are based on experience with SGP4; it is not known whether a larger word or buffers would help in different type applications.

Another recommendation is that floating point hardware be developed and used in those applications that use floating point arithmetic.

On the RADC STARAN, all arithmetic is done in software. This produces long execution times (890 μsecond for a floating point divide) and a large code expansion (75 lines of code for a floating point addition). Fourteen additions would produce over 1,000 words of code. Hardware floating point could have a significant impact on both problems.

The addition of more control memory could make similar programs easier to implement. Typically a large amount of data must be input to and output from a program. In SGP4 there were 16 output variables for each object. Since 512 objects were being processed simultaneously, 8K words of control memory had to be set aside. Because of the software floating point code expansion, the entire program could not fit into control memory and an overlay was needed.

Another important recommendation is the incorporation of a sequential arithmetic capability into AP control. At times, computations are needed that involve just a few numbers. This is true especially when a more general routine is desired. A number may be passed as a parameter to a routine and it may be desirable to work with this number. In the existing system this arithmetic can be accomplished in either of two ways. The operations can be done in the arrays thereby misusing the parallel power of the system, or it can be done on the sequential processor. This involves notifying the STARAN Program Supervisor, transferring the operands to the sequential processor, having the SP do the operation(s) and transferring the results back to AP control memory. This capability would not necessarily be that desirable for other types of programs.

Most of the above recommendations are based mainly on the experiences gained in the experimental implementation of SGP4. Since other types of computations have not been examined, it is impossible to say whether or not these changes would be desirable for other applications. The changes suggested below are believed to be changes which would benefit any user of a STARAN system.

Each page is 512 words long. Larger pages would enable more of a user's program to be run out of high speed semiconductor memory. System software routines such as floating point now occupy a large portion of the three available pages. This effectively eliminates the use of the program pager. The paging mechanism itself could be improved. Segmentation of the program into pages now is under programmer control. He must estimate when his segment is reaching 512 words. Then he must insert assembler instructions to do the actual paging. The assembler could do this segmentation itself and insert the proper instructions. Alternatively hardware could do paging without explicit instructions.

More registers would make programming easier. Additional data pointer (DP) registers would make transfers between control memory and array memory easier. The DP register is used as a pointer into control memory and can be automatically incremented or decremented after instructions. This addition would be especially useful in PIO control where at times the DP register performs other functions. More field pointer (FP) registers would also be useful. They are used to point to bit columns or words in the array. They can also be used to hold parameters when using user-created subroutines.

Another desirable feature would be the speeding up of communications between control memory and array memory. The performance of the RADC configuration is limited by the transfer of data into and out of array memory. This could be speeded up either by using faster control memory or by making a wider data path between control and array memories.

Communications between STARAN and the Honeywell 645 would be speeded up by widening the 12 bit path between the two machines. Transfer to large files have taken up to 20 minutes. All remote users are affected by this time.

SECTION V

SUMMARY AND CONCLUSIONS

Implementation of the SGP4 equations on the RADC STARAN produced a significant increase in throughput. Throughput was heavily dependent on the partitioning of array memory. Because of this dependence, the thorough investigation of the three array allocation schemes was essential. It seems likely that a similar dependence will be found for throughput in similar problems. After array allocation, particular attention should be paid to the array memory layout during processing. This is necessary to make optimum use of the input/output and computational capabilities of the system.

Other than system software development, programming was not a major task in the implementation of SGP4. However, this may not be true of other programs. User-developed system software, if necessary, will probably be a major portion of such an implementation effort. In general, the system software was very good. If a cross assembler is available, assembly time should not be a major concern.

Use of a larger array word has been suggested to help alleviate the input/output problem. This would eliminate the requirement that temporary variables be stored in another word or in control memory. For the type of scientific processing dealt with in SGP4, this is the most important recommendation. Floating point hardware should offer a substantial increase in performance in applications requiring floating point computations. In the SGP4 experiment, throughput could be more than doubled by the use of such hardware. Applications requiring large programs or large amounts of data would benefit from the addition of more control memory.

The experimental implementation of SGP4 on the RADC STARAN system has produced many interesting and useful results. It is hoped that the information gathered can be useful to system designers looking for an improved AP and to present and prospective RADC STARAN users.

APPENDIX A

RADC SYSTEM DESCRIPTION

HARDWARE

The RADC Associative Processor (RADCAP) Testbed Facility consists of a Goodyear Aerospace Corporation STARAN S-1000 associative processor interfaced to the HIS 645 Multics system. An overview of the system appears in Figure 1. The associative processor can be operated in two modes, a stand-alone mode and an on-line mode to the Multics time-sharing system. In the latter mode, a Multics user is able to control the STARAN from his terminal as he would if he were using the STARAN in stand-alone mode. He can create program and data files using the capabilities of Multics and transmit them to STARAN. Currently the associative processor cannot be time shared; that is, only one user at a time may utilize the STARAN. All communications between STARAN and Multics are via a 12-bit parallel buffered I/O channel.[4]

The RADCAP STARAN basically consists of a conventionally addressed control memory for program storage and data buffering, four associative memory arrays, a control logic unit for sequencing and decoding instructions from control memory, and a control logic unit associated with a special parallel input/output (PIO) capability.[5] A typical STARAN associative array memory module is shown in Figure 2. The associative array memories are the heart of the STARAN system. The array memories provide content-addressability and parallel processing capabilities. Each array consists of 65,536 bits of multidimensional access (MDA) memory organized as a memory matrix of 256 words by 256 bits with parallel access to up to 256 bits at a time in either word (horizontal) direction, bit-slice (vertical) mode or mixed mode. Mixed mode allows access to 2^n bits from each of 2^{8-n} memory words. For example, one byte (8 bits) can be taken from each

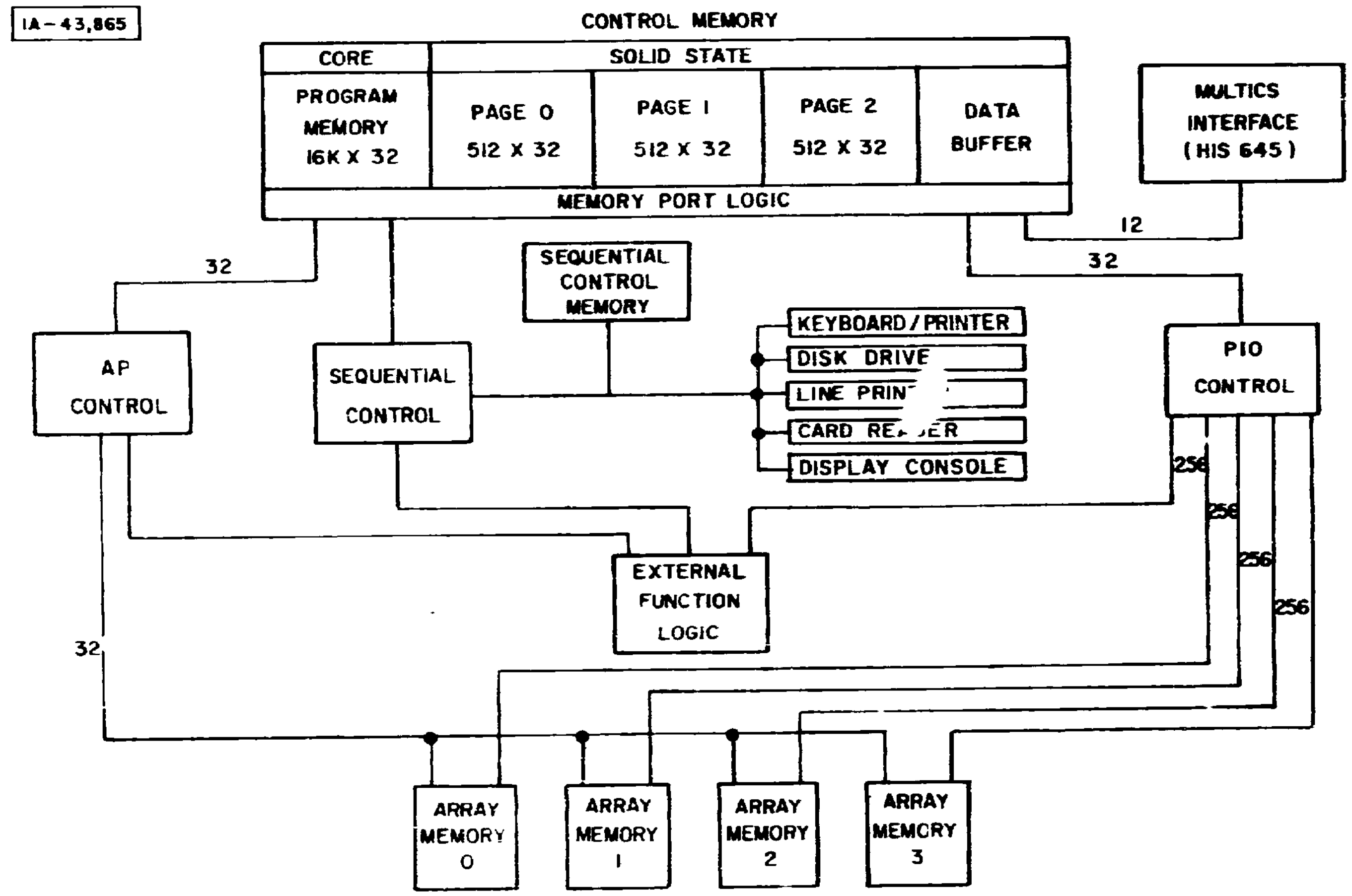

Figure 1 OVERVIEW OF RADC STARAN SYSTEM

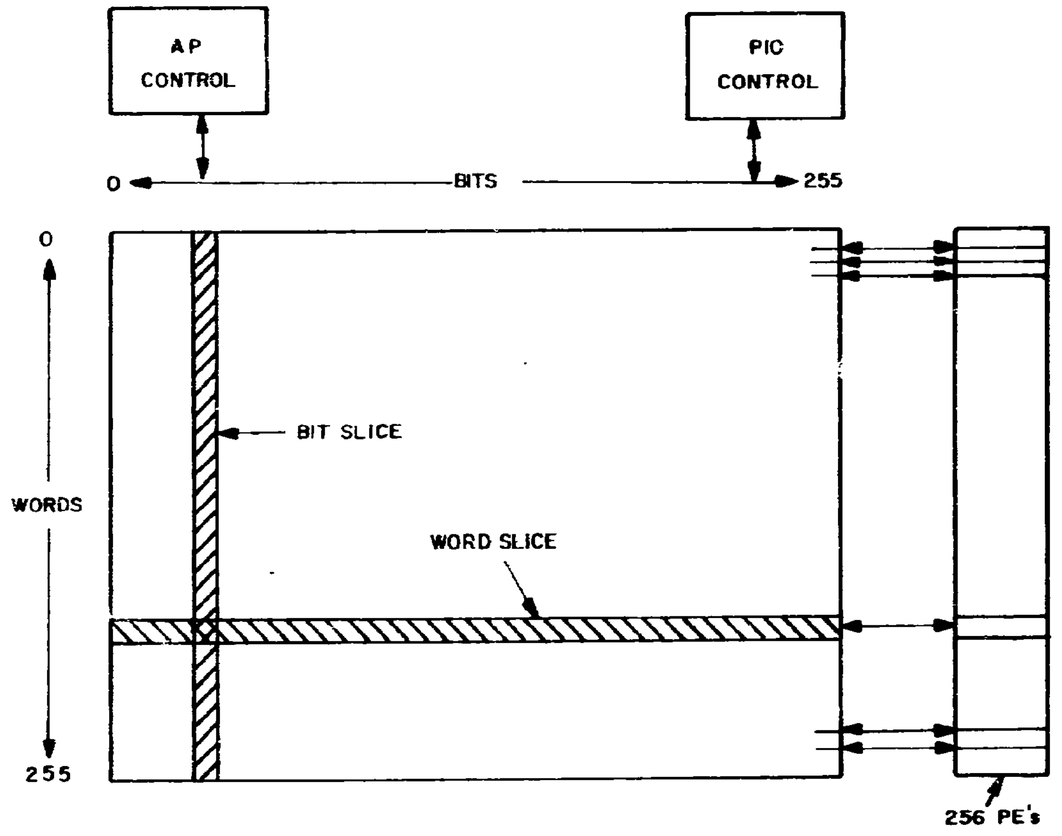

Figure 2 STARAN ARRAY

of eight words. In addition to the MDA memory, each array contains 256 bit-serial processing elements. Each of the 256 processing elements consists of three bits of response store and associated logic. These 256 bit-serial processing elements provide the parallel processing capabilities for each array. Each PE can be considered a simple 1-bit miniprocessor with an associated memory of 256 bits and three 1-bit registers. In the RADC configuration of four arrays there are 1024 of these miniprocessors. Arrays can be under the control of either the AP control logic unit or the PIO control logic unit. In order to perform arithmetic or logical operations on the arrays, all arrays would be under the control of AP control. Thus, all 1024 processing elements would be affected. Instructions are sequenced and decoded by the control logic and executed simultaneously by all active processing elements. This allows bit-serial arithmetic operations or bit-serial search operations to be performed on all words of memory for all arrays in parallel. For example, if field A was defined to be bits 0-31 of each word and field B bits 32-63, field A could be added to field B simultaneously in all words of memory. Also field A could be searched for a particular value simultaneously in all words of memory. Processing in the STARAN system can be overlapped with some arrays performing input/output while others are executing arithmetic and logic instructions. This is done by assigning individual arrays to either AP control or PIO control.

Other major elements of the RADC STARAN system are explained below.

AP Control Memory

The functions of the AP control memory are to store the AP application programs and to hold data for those programs. Since AP control memory is not directly tied to the array memory, AP control can overlap the AP control memory cycle time with the array memory cycle time.

Control memory is divided into several memory sections. Three fast semiconductor page memories can contain the current AP application program segment; the slower core memory can contain the rest of the AP program. Each of the pages contain 512 32-bit words. Page zero is typically used to contain a library of system routines such as arithmetic subroutines. A separately controlled program pager can be used to bring in segments from slow memory to the pages. Pages one and two are used in flip-flop 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 data buffer, like the page memories, is a semiconductor memory. It also contains 512 32-bit words. Except for the pager bus, all buses that can access the AP control memory can also access the data buffer to store data or fetch instructions that need to be accessed quickly by the different elements of STARAN.

Bulk core memory uses standard core storage. The RADC configuration contains 16K 32-bit words. The main use of the bulk core memory is to store AP application programs. Since the bulk core memory is accessible to all buses it is also used for data storage for items that do not require the high speed of the data buffer.

PIO control memory is an area of 512 32-bit words of semiconductor memory. It is from this area that the PIO control fetches its instructions. It can also be used to hold instructions or data for AP control.

AP Control Logic

The primary function of the AP control logic is to control operation of the STARAN associative array memories. All arithmetic and logical instructions are executed within the arrays under control of AP control. A 32-bit data communications path between control memory and the arrays is also provided.

Program Pager Logic

The program pager loads the fast page memories from the slower bulk core memory. While the AP control logic is executing a program segment out of one page, the program pager can be loading the other page with a future program segment. Whether a program is to be executed from bulk core memory or paged to the page memories and then executed is entirely under programmer control.

External Function Logic

External function logic enables the AP control, sequential control, or an external device to control the operation of STARAN. By issuing external function codes to the external function logic a STARAN element or external element can interrogate and control the status of the other elements. The external function logic also provides a means of communication between a STARAN operator and STARAN via function switches on the STARAN console.

Sequential Control Processor

The sequential control portion of STARAN consists of a PDP-11/20 minicomputer with 8K 16-bit words of memory, a keyboard-printer, a CRT alphanumeric display, line printer, card reader, perforated tape reader/punch unit and cartridge-type disk memory. The sequential processor also contains logic to interface with other STARAN elements. It runs system software programs such as the assembler and macro preprocessor, operating system, file handling programs, diagnostic programs and debugging routines.

Input/Output Options

The custom input/output unit (CIOU) provides STARAN with several extra I/O functions. The parallel input/output option provides an I/O channel up to 256 bits wide for inter-array communications. It also gives an additional 32-bit data path between control memory and associative memory. All of the PIO functions can be executed at the same time that AP control is executing. Each array can be assigned to be either under the control of AP control or of PIO control. The custom interface unit implements a buffered input/output channel for external communications with the Multics system or with STARAN peripherals in a stand-alone mode of operation. The parallel input output channel is implemented to provide rapid inter-array communications and to provide an interface to any future high-bandwidth external storage devices such as parallel head-per-track disc systems. A hardware performance monitor is also supplied with the CIOU. The monitor has an event counter and an elapsed time counter that is accurate to one hundred nanoseconds. Control of the monitor is from within a user program.

SOFTWARE

The programming language used on the STARAN is an assembly language called APPLE (<u>A</u>ssociative <u>P</u>rocessor <u>P</u>rogramming <u>L</u>anguag<u>E</u>). Unlike most assembly languages, some APPLE mnemonics generate more than one machine language instruction. These mnemonics are generally the associative instructions which do arithmetic or searching operations on the arrays. The types of the instructions are as follows:[6]

1. Assembler directives
2. Register loads and stores
3. Branch instructions
4. Associative instructions
   a) Loads
   b) Stores
   c) Parallel array searches
   d) Parallel array moves
   e) Parallel array arithmetic operations

5. Control and test instructions

6. Input/output instructions

Arithmetic operations can use a value in a 32-bit register as one operand and a value in a field of every array word as the other operand. Another way of doing the arithmetic is to select two fields of every word as the operands. Add, subtract, multiply, divide, and square root are supported by APPLE. The assembler also has a conditional assembly capability. Output is in the form of relocatable or absolute program segments. Relocatable code gives the user the flexibility to combine his program with other programs without worrying about control memory storage allocation.

A macro pre-processor called MAPPLE (Macro APPLE) is also available.[7] It contains arithmetic, logical and string manipulation capabilities. These provide the capability of defining user-defined mnemonics. All floating point instructions are implemented as macros. Since no higher level language exists, all programs must be written in a combination of MAPPLE, APPLE and machine language.

The APPLE Linker (ALINK) is a processing program that prepares the output of the APPLE assembler (i.e., object modules) for loading and execution. It combines object modules, relocates them, then assigns absolute addresses. It resolves external references, generates overlay structures on request, creates a load map containing the absolute addresses of the load module and the entry point(s) of each object module, and produces executable code (a STARAN load module) in format suitable for subsequent loading into storage for execution.

The STARAN Program Supervisor (SPS) is the software interface between the sequential controller (PDP-11) and the associative processor control. This program allows a user to communicate with the sequential controller from an AP program and vice versa.

The STARAN Control Module (SCM) provides operator control of STARAN by means of a system console. Its main function is to load AP control memory (page memory, data buffer and bulk core) and parallel I/O control memory with instructions or data from a file created by the APPLE Linker. It allows a user to control the execution of a program by starting, continuing and halting instructions in either AP control, PIO control or the pager. It also allows insertion of a breakpoint in an AP program and allows communication with any element of STARAN by the issuing of External Function Commands.

STARAN Debug Module (SDM) is a set of commands that allows a user to interactively debug his program. There exist routines to examine and change any memory location including array memory. Dumps can be made on any desired device. A preselected table of memory locations and/or registers can be printed on command. In addition, the STARAN's status can be checked by displaying the status of certain areas on a set of console lights. The performance monitor can also be read and reset.

APPENDIX B

DEFINITION OF PIO ROUTINES

Special PIO routines were developed to take advantage of the dual control feature of STARAN. Two different categories of routines were involved. The first involves data movement between the computation arrays (0 and 1) and the buffer arrays (2 and 3). The other routine moves data between control memory and the buffer arrays. All of the operations are word-oriented; that is, 512 32-bit word transfers are involved in every operation. This is because all data is in the form of 32-bit floating point numbers. Because two arrays are used as buffers and two arrays are used for computation, transfers take place in blocks of two arrays (512 words).

Input and output between the arrays and control memory is done through Port 7 of the PIO control registers. Only 32 bits may pass through this port at once, thus 512 load and 512 store operations are required.

Transfers between arrays are done simply and quickly. Movement can take place 1024 bits in parallel. Three operations are repeated 32 times to accomplish the transfer. First a bit slice from two arrays is loaded into the PIO buffer registers. Then the registers are exchanged with the two other PIO buffer registers. Finally the bit slice from the buffer registers is written back into the two new arrays.

Once the information is in the arrays it is sometimes necessary to transfer it to control storage. This occurs several times when evaluating long expressions with many intermediate, temporary results. In these cases it is possible to reduce the time necessary to transfer data from array memory to control memory. Instead of using 512 load and 512 store operations it is possible to use 32 loads and 512 stores. This is done by writing the data from the array into the buffer register by bit slice rather than by word slice. The data as it appears in control memory is scrambled; that is, the first 256 bits stored into control memory are all from the most significant bit of the indicated word column in the buffer array. This disregard for the appearance of data within control memory allows the increase in speed of the operation.

The routines actually execute through PIO control from PIO control memory. In order to execute them, a calling sequence had to be established from AP control. All of the commands have a similar format:

```
LABEL      OPERATION                  SOURCE, DESTINATION, CHAIN
```

LABEL	-	an optional label can appear
OPERATION	-	specifies the specific operation desired
SOURCE	-	where the data is to come from, either control memory address or bit column
DESTINATION	-	where the data is to go, either control memory address or bit column
CHAIN	-	a parameter to allow command chaining
	-	null indicates no chaining
	-	0 indicates same operation to follow
	-	1 indicates similar operation to follow

There are two groups of similar operations; the first group is the transfers between computation and buffer arrays and the other is the transfers into and out of control memory.

Each command is expanded as a macro. The first instructions store the address of the in-line parameter list in a location accessible to the PIO routines. A branch is then coded to bypass the parameters. The parameters contain the address of the routine desired, the source address, the destination address and the chaining indicator. If no chaining is indicated, that is the end of the parameter list.

If same operation chaining is indicated, the parameter list is repeated with the same format but with the routine address eliminated. If similar operation chaining is indicated, the parameter list is repeated, but with the address of the appropriate new routine preceding it. Both types of chaining can be extended as many times as desired. Without chaining, overlap of execution between AP and PIO control is virtually impossible.

AP control must initiate all activity in PIO control. Without chaining, each I/O instruction would have to be started up individually. While one I/O instruction is executing the next I/O instruction would wait until PIO control is free. AP control would do this checking and so would not be able to execute other instructions. With chaining the I/O instructions can be queued in a parameter list. AP control starts up PIO control and is then free to execute other non-I/O instructions. This minimizes the amount of communication needed between AP and PIO control. Once PIO control is started it can get all the information it needs from the parameter list. After the parameter list are two instructions to test the PIO unit to see if it is running. If it is, th machine waits for this activity to halt. This ensures that a currently active PIO routine will not be interrupted. Next, if the operation is to involve transfers between the arrays, arrays 0 and 1 are assigned to PIO control. Under all circumstances arrays 2 and 3 are under PIO control. Next, for both types of operations, PIO control is enabled. The address of the parameter list is picked up and PIO control begins executing. In AP control, if the operations do not affect arrays 0 and 1 the macros are completed and the next instruction appearing after the PIO command is executed. If the operations do affect arrays 0 and 1 (interarray transfers) there is a wait until PIO control finishes executing. Arrays 0 and 1 are then assigned back to AP control.

REFERENCES

[1] Lane, M., and Cranford, K., "An Improved Analytical Drag Theory for the Artificial Satellite Problem," AIAA Paper No. 69-925, 1969.

[2] Kozai, Y., "The Motion of a Close Earth Satellite," The Astronomical Journal, Vol. 64, No. 1274, 1959.

[3] Brouwer, D., "Solution of the Problem of Artificial Satellite Theory Without Drag," The Astronomical Journal, Vol. 64, No. 1274, 1959.

[4] STARAN/HIS-645 Users Guide, GER-15641, Goodyear Aerospace Corporation, August 1973.

[5] STARAN Users Guide, GER-15644, Goodyear Aerospace Corporation, August 1973.

[6] STARAN S Apple Programming Manual, GER-15637A, Goodyear Aerospace Corporation, August 1973.

[7] STARAN S Macro Programming Manual, GER-15643, Goodyear Aerospace Corporation, August 1973.