The IBM 2938 *Array Processor discussed enhances the processing power of* SYSTEM/360 *Models 44, 65, and 75 for operations on vectors and matrices. As an integrated channel-*I/O *device, the array processor responds to standard* I/O *instructions.*

Discussing the overall flow of control between the central processing unit, main storage, and the array processor, the relationship between the SYSTEM/360 *Operating System and the array processor's programming support is emphasized. The twelve mathematical processing operations embodied in the* 2938 *are described in terms of their algorithms.*

Several methods for measuring array processor performance are shown together with actual timing results.

An auxiliary processing system for array calculations

by J. F. Ruggiero and D. A. Coryell

The value of a computer instruction operating upon multiple data elements was recognized at least as early as 1953 when Andrew and Kathleen Booth were designing computers in London. They mention the capability of "addition of a series of pairs of numbers stored consecutively" and forming, by means of branching and indexing, "the total of a series of consecutive numbers," each under control of a single instruction.[1] The advantage of such multiple operations lies in the possibility of more efficient utilization of main storage accesses by reducing the number of instructions fetched during the operation.

The IBM 2938 Array Processor extends the concept of multiple operations. The processor adds auxiliary computing power to the Central Processing Units (CPU's) of certain IBM SYSTEM/360 models when these are performing operations on arrays (vectors and matrices) and when calculating convolutions and correlations.[2] This auxiliary processor is most advantageously applied where the ratio of calculation time to input/output (I/O) time is relatively high, as in the convolution of two arrays.[3]

The operations apply particularly well to digital signal processing and filtering. For example, the reduction of seismic data may involve from two million to 20 million multiply-add calculations per output record. However, the array processor is not restricted to a particular application area since the operations provided are widely used mathematical tools.

Basic vector operations (element-by-element additions and multiplications) are useful in many application areas. One such example is that of making weather forecasting calculations. Matrix multiplication and vector inner-product operations, though less broadly applicable, are also useful in such areas as structural stress analysis, electrical network solutions, and probability analysis of variance. The convolution/correlation- and difference-equation operations apply to digital signal processing, which in turn is a useful tool in many areas since the signals being processed may arise from many types of systems—mechanical, electrical, or optical.

This paper discusses first the array processor as an I/O device that can be attached to three models of SYSTEM/360. Similarities and differences between the array processor and other I/O devices are described. Discussing the overall flow of control between the CPU, main storage, and the array processor, the relationship between the SYSTEM/360 Operating System (OS/360) and the Array Processor Access Method (APAM) is emphasized.[4] The twelve mathematical operations embodied in the array processor are presented. A discussion of APAM stresses methods for concurrent operations of the CPU and the array processor. Computation efficiency is demonstrated by tables of array processing timing data.

The array processor

The 2938 Array Processor is an integrated channel-I/O device designed to enhance the performance of SYSTEM/360 Models 44, 65, and 75 by providing peripheral processing capabilities for the system to which it is attached. The processor is connected to the CPU as a channel, as shown in Figure 1. Communication between the CPU and the 2938 is conducted in the normal SYSTEM/360 I/O

Figure 1 Array processor-SYSTEM/360 configuration

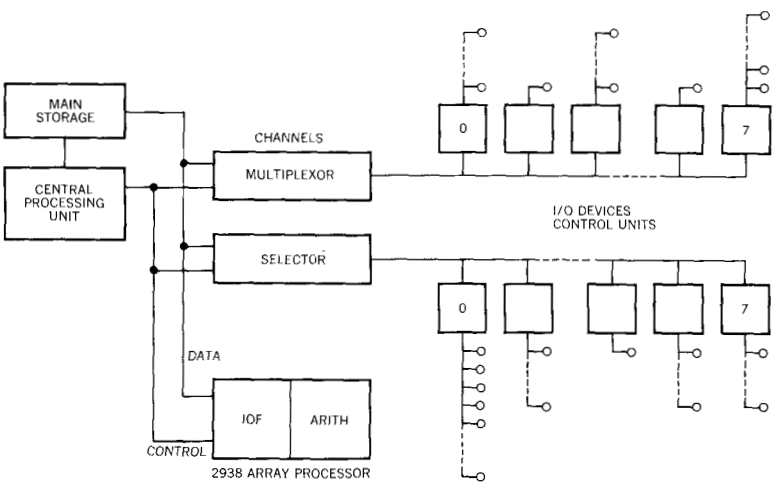
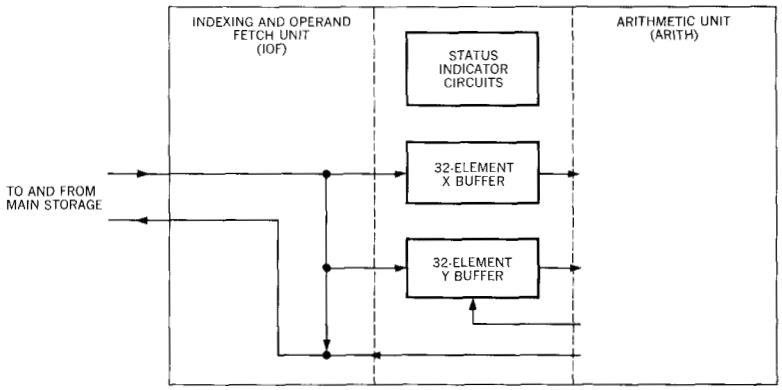


Figure 2 The IBM 2938 Array Processor



manner.[5] Thus, the array processor responds to standard I/O instructions and uses the channel address word (CAW), channel command word (CCW), and channel status word (CSW). When operations are initiated by a START I/O (SIO) instruction, the array processor fetches input data from main storage, performs the desired operations, and returns the results to main storage. The processor gains access to data in main storage by sharing cycles with the CPU and the other channels. Control flow for these operations is governed by a CCW list in main storage.

The 2938 is designed to perform twelve array and matrix operations in addition to NO OPERATION (NOP), SENSE, and TRANSFER IN CHANNEL (TIC).[6] These operations are implemented in two independent microprogrammed units within the array processor. Figure 2 shows these two units in greater detail. The Indexing and Operand Fetch (IOF) unit fetches input data from main storage and returns calculated data to main storage. Calculations are performed in the Arithmetic (ARITH) unit. The IOF unit operates asynchronously with the ARITH unit, and it also performs fixed-to-floating point number conversion as required.

The IOF (and therefore the array processor) accepts data in the SYSTEM/360 short floating-point format and in the half-word, fixed-point format.[6] Also, the 2938 recognizes a nonstandard SYSTEM/360 fixed-point format. This is called the sign-magnitude format, which consists of fifteen bits plus a sign. The fifteen bits are treated as an absolute binary number. The sign bit (bit 0) indicates positive or negative values. This special format is frequently encountered in analog-to-digital or digital-to-analog conversion equipment. By having the ability to accept input data in this format, data conversion between complement and sign-magnitude representations can be eliminated, resulting in significant time savings.

When the array processor is performing calculations, the ARITH and IOF units communicate by means of two 32-element internal buffers and status indicator circuits. The arithmetic unit

is essentially a high-speed, floating-point, four-stage multiply-add unit (i.e., a multiply-align-add, post-normalize unit). The two 32-element internal buffers are used to supply data to this high-speed arithmetic unit, which has a maximum processing rate of 200 nanoseconds per stage. Thus, depending upon the operation, product sums can be formed at a rate up to 5,000,000 per second. All arithmetic operations are performed using the SYSTEM/360 short-format floating point representation.

A number of I/O devices may be operating simultaneously with the 2938. Since the array processor operates at higher rates than other I/O devices, it is normally assigned a priority lower than that of the other channels. This assignment assures that overrunable I/O devices are not held up or forced into an overrun condition by the 2938. If necessary, the array processor waits for a requested access to main storage, after the completion of internally overlapped processing, until the request is granted.

System operations

We now discuss the execution of an application program using the Array Processor Access Method (APAM) and the SYSTEM/360 Operating System (OS/360) or the Model 44 Programming System (44PS). Figure 3 illustrates an application program requesting array-processor facilities. A call to APAM initiates the dynamic construction of the necessary channel program. APAM then determines if the array processor is responding to a previous request. If the array processor is busy, the new request is placed in an *output-restricted deque*.[7] (The phrase output-restricted deque, pronounced deck, is defined as a list wherein requests may be entered at either end, but may be removed only from the front of the list.) If the array processor is available, control passes to the operating system control program via an Execute Channel Program (EXCP) request. The control program issues a START I/O (SIO) instruction to the array processor. The normal I/O functions follow, up to and including the fetching of a CCW. Each CCW in a channel program can specify one of the array operations that are described later in this paper. (After the SIO instruction is issued, the array processor operates in parallel with the CPU.)

An extension to the SYSTEM/360 I/O procedure is required to provide sufficient flexibility in the specification of these array operations. The added flexibility is accomplished through the introduction of a new control word, the Operand Control Word (OCW). Each CCW may point to an OCW list, which provides details concerning format, size, and main storage addresses of the arrays of data involved in a particular operation. The array operations may involve two or three arrays of data elements (fixed-point or floating-point numbers), resident in main storage. Array processor operations on these arrays are concurrent with other CPU activities.

Figure 3 An application program using array processor facilities

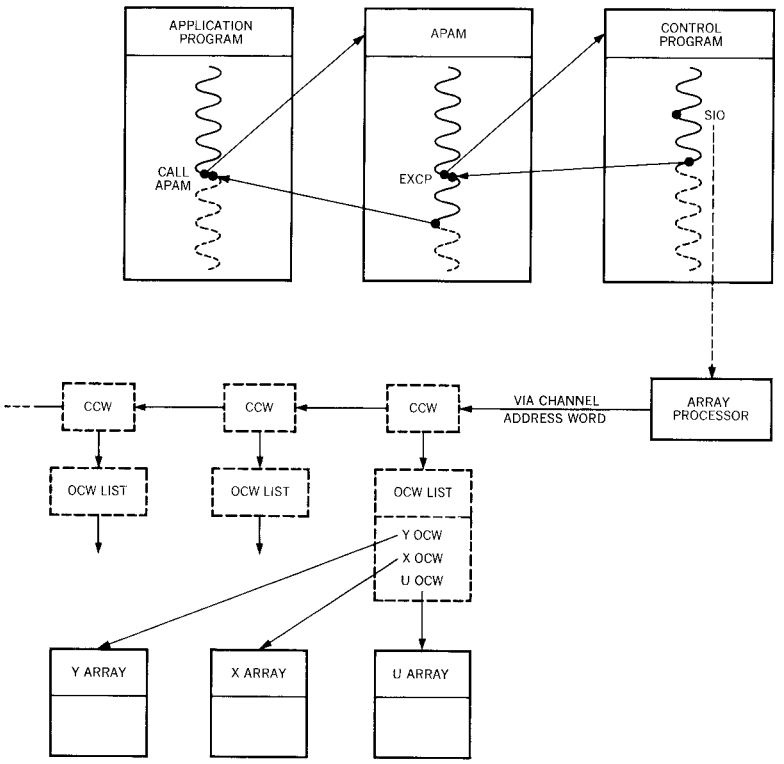
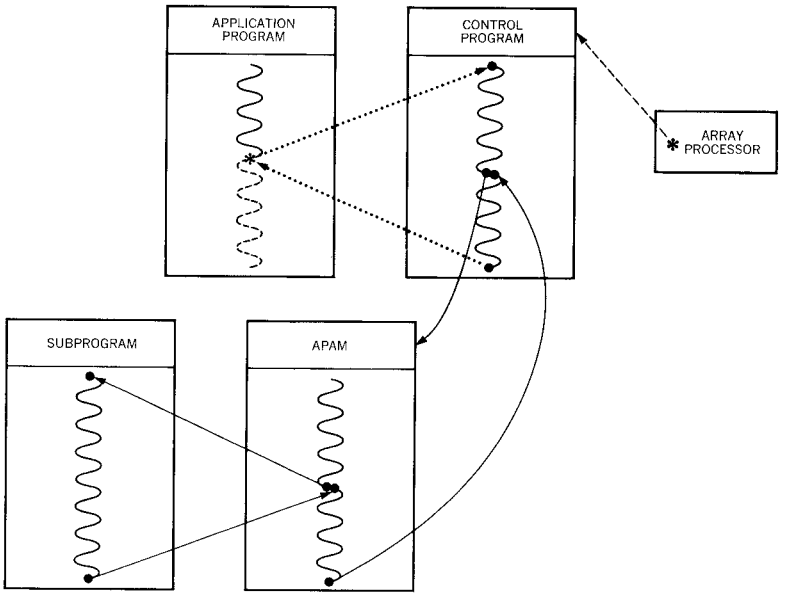


Figure 4 Processing an array-processor interruption



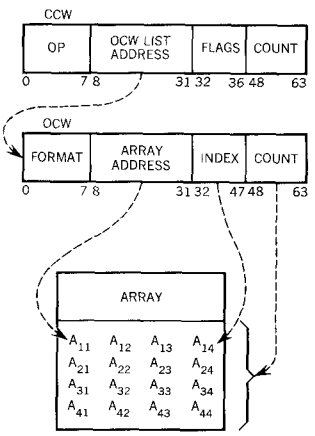
At the completion of the operation designated by a CCW, the command-chaining bit is inspected. If the bit is "on," the next CCW (which may specify a completely different array operation) is fetched, and array processor operations continue. The CCW list is exhausted when a CCW is fetched whose command-chaining bit is "off." The 2938 then signals I/O completion via an I/O interruption.

Figure 4 illustrates a 2938 interruption (asterisk) occurring during the execution of an application program. However, interruptions are permitted whenever APAM, the control program, the application program, or an unrelated program is in control. Here, the control program passes control to an APAM routine after saving the system status information. APAM then does some standard interruption processing and links to a user-supplied subprogram if the user desires control on this type of interruption. (The user may issue additional requests to APAM from within his subprogram.) Control is returned to APAM, which restarts the 2938 by issuing an EXCP for the request at the head of the request deque. Control is then returned to the supervisor where the control program completes its interruption processing and returns control to the original point of interruption in the application program.

CCW format

The formats of the CCW and OCW as well as the interrelationships between these control words are shown in Figure 5. (Bit positions 37 to 47 of the CCW are unused.) Referring to the CCW, the operation field (OP) specifies the particular array operation to be performed. The OCW list address specifies the location of the OCW list. The count field specifies the length of the OCW list in bytes. Flags interpreted by the 2938 are: command chaining flag, suppress incorrect length indication, and Program Controlled Interruption (PCI).

Figure 5 Formats of the CCW and OCW related to an array



Of the three, the PCI is of particular interest. The PCI facility, as implemented on the array processor, offers synchronization capabilities not available on standard SYSTEM/360 models. For example, when a CCW is fetched, the PCI bit is interrogated. If the bit is "on," array processor operation is suspended after the fetching of the associated OCW's, if any, until the interruption is accepted by the CPU. Thus, if array processor interruptions are disabled, i.e., the bit in the system mask byte of the Program Status Word (PSW) that corresponds to the channel is set to zero, the array processor waits until the channel is enabled before processing continues. This operation guarantees one I/O interruption for every PCI. Referring again to the CCW, if the PCI bit is "off," the array processor operates in the normal manner.

OCW format

We now refer to the OCW format, also shown in Figure 5. The OCW's are double words that contain the following four information fields:

- Format field
- Address of the initial array element to be fetched or stored

- Two's-complement index value that permits forward (increasing address) or reverse indexing of the array address for the second and subsequent accesses
- Count of the number of elements in the array

The format field provides control of the format converter in the Index and Operand Fetch (IOF) unit of the array processor. Within this field may be specified whether the data is in fixed-point or floating-point form—and, if in fixed-point form, whether signed or two's-complement notation is used. The algebraic sign of the elements in the array may also be removed or inverted while fetching the elements. This facility can be considered as an operation-modifier field. Thus, without rewriting an array or changing the data in main storage, the array processor may use the negative, absolute value, or the negative of the absolute value of each data element, as well as the algebraic value.

A bit in the format field of the OCW controls the result field. This bit may be set "on" to indicate that the previous contents of the result field are to be fetched and added to the results being generated by the current operation. (This function is called "stacking" by analogy with the signal-processing technique of averaging out random fluctuations by combining several similar signals.)

Note that each array involved in an operation is specified by a different OCW. Therefore, arrays with different attributes can be used in the same operation.

Table 1 Vector move/convert (VMC)

$Y_i = X_i$
for
$i = 1$ to N
where
$N = \min$ (CTY, CTX)

Table 2 Vector-float-to-fixed-point convert (VFX)

$Y_i = X_i$
for
$i = 1$ to N
where
$N = \min$ (CTY, CTX)

Array processing operations

Thus far, we have discussed interactions between the CPU and array processor. We now explain the array processing operations in terms of their algorithms as implemented in the 2938. In these algorithms, the following conventions are used: Y represents the resultant array, X is the first source array, and U is the second source array. For each algorithm, we give the operation name and its mnemonic representation as well as a brief description of the operation. Algebraic statements of the algorithms are displayed in table form.

As previously discussed, the operation field of each CCW specifies the operation desired, and the corresponding OCW specifies the attributes of that operation. Thus, in effect, the CCW's are collectively the instruction set of the array processor. Format conversion and sign-modification operations, previously discussed, apply to each of the operations of the 2938 array processor, with only a few exceptions, which are noted where they apply. Also, the forward or reverse indexing of each array and the count field for each array provide for some modification of the operations. These, too, are noted where they apply. References 4 and 8 give detailed information on the programming support and array-processor implementation of the operations.

Table 3 Scalar multiply (SMY)

$Y_i' = Y_i + X_i U$
for
$i = 1$ to N
where
$N = \min$ (CTY, CTX)

Vector move/convert (VMC) is a simple relocation of data within storage with some added capability (see Table 1). The format converter permits the changing of fixed-point data to floating-point data during relocation. Sign control permits the generation of complement, absolute, or complement of absolute (all negative) array elements. Independent indexing of source and resultant arrays permits the reversing of the sequence of elements in an array or the writing of the transposition of a given matrix, i.e., $a_{ji} = b_{ij}$. The separate count fields may be used to fill a resultant array field with zeros after the count of a source array field is exhausted. A character field may be moved if it starts and ends on suitable boundaries. A maximum of 65,535 elements may be moved in a single execution of the VMC instruction, and each element may be up to four bytes long. The terms CTY and CTX represent the counts (numbers) of elements in the Y and X arrays, and N is equal to the lesser ("min") of the two counts.

Vector-float-to-fixed-point convert (VFX), changes a source floating-point array to a resultant fixed-point array (Table 2). Because the set of possible floating-point numbers is greater than the set of possible fixed-point numbers, the conversion also involves scaling. The characteristic of each floating-point source element is compared to a maximum-value element. The fraction portion of the source element is shifted right until the characteristic of the source element equals the maximum-value element. A scalar multiply is then performed on the adjusted source element.

Scalar multiply (SMY) causes each element of the X source array to be multiplied by a single element furnished in the U source array (Table 3). The resultant product element, may be added to the contents of the Y array.

Vector element-by-element sum (VES) performs the addition of corresponding elements in each of the two source arrays and places the sums in a result array (Table 4). A control bit in the OCW format field for the U array permits a single U element to be added to each element in the X array.

Vector element-by-element multiply (VEM) performs the multiplication of corresponding elements in each of the two source arrays and places the products in a result array (Table 5). Optionally, the resultant product elements may be added to the contents of the Y array.

Sum of vector elements (SVE) performs (see Table 6) the summation of all elements of a source array and stores the sum (a single resultant element) in a specified location. Optionally, the resultant element can be added to the contents of a storage location to produce a new resultant element.

Sum of squares (SSQ) is similar to SVE except that each element is multiplied by itself and the squares of the elements are summed (Table 7). The sign of each element may be applied to each of the products, or a positive value may be taken by setting the absolute-value control bit in the X format field of the OCW. This op-

Table 4 Vector element-by-element sum (VES)

$Y_i = U_i + X_i$
for
$i = 1$ to N
where
$N = \min$ (CTY, CTX)

Table 5 Vector element-by-element multiply (VEM)

$Y_i' = Y_i + U_i X_i$
for
$i = 1$ to N
where
$N = \min$ (CTY, CTX, CTU)

Table 6 Sum of vector elements (SVE)

$Y' = Y + \Sigma_{i=1}^{N} X_i$
where
N = CTX

Table 7 Sum of squares (SSQ)

$Y' = Y + \Sigma_{i=1}^{N} X_i|X_i|$
where
N = CTX

Table 8 Vector inner product (VIP)

$Y' = Y + \Sigma_{i=1}^{N} (U_i X_i)$
where
N = min (CTX, CTU)

Table 9 Signed, squared array (SSA)

$Y_i' = Y_i + X_i|X_i|$
for
$i = 1$ to N
where
N = min (CTY, CTX)

Table 10 Partial matrix multiply (PMM)

$Y_i' = Y_i + \Sigma_{j=1}^{N} (X_j U_{(i-1)n+j})$
for
$i = 1$ to N
where
n = CTY
N = CTX

Table 11 Convolving multiply (CVM)

$Y_i' = Y_i + \Sigma_{j=1}^{N} U_j X_{i+j-1}$
where
$i = 1$ to m
and
m = CTY
N = CTU

Table 12 Convolving addition (CVA)

$Y_i' = Y_i + \Sigma_{j=1}^{N} |X_{i+j-1} + U_j|$
for
$i = 1$ to m
where
m = CTY
and
N = min (CTX, CTU)

eration produces a single-element result that may (optionally) be added to the contents of a storage location to produce a new resultant element.

Vector inner product (VIP) is the formation of a single result from two input arrays, namely, the sum of products of the elements of the two input arrays (Table 8). Mathematically, the operation corresponds to the premultiplication of a column vector by a row vector. The resultant sum may either be stored or added to the contents of a storage location.

Signed, squared array (SSA) performs the operation of squaring each element of the input array and storing the resultant array, retaining the signs of the original source array elements if desired (Table 9). Optionally, instead of restoring the resultant elements, they may be added to the corresponding element locations in storage to form new resultant elements.

Partial matrix multiply (PMM) is an expansion of VIP, wherein elements of a (resultant row) vector are calculated which are products of elements of a row vector premultiplying a source matrix (whose elements are arranged according to the FORTRAN convention for matrix storage). The corresponding algebraic statement is shown in Table 10. The resultant row vector may be stored or added to a row vector in storage.

Convolving multiply (CVM) performs the partial correlation of two source arrays, starting with the one-for-one alignment of the first element in each array (Table 11). By specifying the high address of either array as the starting address and using a negative index value, the user may effect a convolution of the two arrays. The resultant convolution elements may be stored or, optionally, added to an array in storage.

Convolving addition (CVA) operates similarly to CVM except that the sum of sums is calculated instead of the sum of products (Table 12). The ending conditions correspond to those for CVM, taking into account the difference in identity elements for addition and multiplication.

High-value readout is an operation that is normally command-chained to a previously given control operation. Upon completion of an array operation, the sign, characteristic, and high-order hexadecimal digit of the result element having the highest value is retained in the array processor. This value may be retrieved by the high-value readout operation and may then be used as the maximum-value element for a VFX instruction.

Sense is the standard error sensing operation for all I/O devices; differences in function are related to the inherent differences in the I/O devices themselves. The array processor uses two status bytes for error sensing. Of special interest are the floating-point overflow and the halt-I/O indicator bits. Floating-point overflow is indicated when a result having a maximum magnitude representable in floating-point notation is returned to main storage. A floating-point overflow is also indicated during a VFX if the input floating-point element is greater than the maximum-

value element. The halt-I/O indicator is set when an array-processing operation is terminated by a HALT I/O instruction.

Programming support

The Array Processor Access Method (APAM) provides the normal programming functions of an access method in an interpretive mode. APAM also has some special I/O-handling features such as concurrently accommodating queued- and EXCP-levels of access to the 2938. The access method is usable with both FORTRAN and assembler-language programs.

I/O overlapping

The overlapping of CPU operations and array processing is an I/O capability not normally available to FORTRAN programmers.[9] Normally, when a FORTRAN application program issues a request for I/O service, the program does not receive control until the I/O operation is completed. Thus, the program must wait, even if there is some interim processing that can be done while the data transfer associated with the I/O request is taking place. APAM, however, allows operations either in this normal manner or in an overlapped manner. To overlap CPU and array processor operations, the application program may request that control be returned immediately via one of the parameters associated with the CALL statement to APAM. The user program may then perform additional processing. The overlapping procedure is the recommended one since it is one way to force the concurrent operation of the array processor and the CPU. While operations overlap, additional requests may also be issued to APAM by the application program.

interpretive capability

Although control operations are normally used for device setup and error recovery, the array processor's major functions are implemented in the form of control operations. Due to the great variety of channel programs possible, it would be very costly to have all channel programs prebuilt at compilation time, especially when there is a possibility that many of them would never be executed. Channel programs may also be data dependent as they are in certain data-reduction applications. Thus, an interpretive I/O capability at execution time, which facilitates APAM's "build only" option, enables the application program to request APAM to construct a channel program at execution time. With this option, the application program may have a channel program built and saved as well as built and executed. The user may then use an EXCP-level of interface with APAM and pass the channel program as a parameter. This capability offers a means of eliminating unnecessary channel program construction for repetitive operations. The high-value readout command is supported in the interpretive mode, whereby the user may request the saving of the maximum exponent after any matrix operation.

APAM capabilities

When the 2938 is attached to a SYSTEM/360 Model 44, the user interface consists of the Model 44 version of APAM and the

Model 44 Programming System (44PS). When the 2938 is attached to SYSTEM/360 Models 65 or 75, the user interface consists of APAM and OS/360.

The user specifies array processor operations to APAM by means of FORTRAN CALL statements. At both the basic and queued levels, APAM constructs the necessary channel programs and requests the control program to initiate 2938 operations. The application program may wait for completion of the operation or may request that control be returned immediately (so as to perform additional processing concurrently with the array processor) as previously discussed. At the EXCP level, APAM has the ability to execute a user-built channel program. If this program is made up completely or in part of available APAM options, the user may have APAM's "build-only option" construct and save the channel program. By means of this option, channel programs of varying length are dynamically created and saved at execution time.

APAM provides a synchronous user exit whenever the control program detects a Program Controlled Interruption (PCI) from the array processor. The user may synchronize the 2938 with the rest of the system by use of the PCI facility in the following manner. At any time prior to issuing a request that causes a CCW (with its PCI bit set) to be executed, the application program may issue a special request to APAM. Such a request specifies the entry point of an application subprogram. Any PCI encountered thereafter causes the subprogram to be entered. Upon exit from the subprogram, the application program may request a HALT I/O instruction to be issued to the 2938.

Table 13 An example channel program

Main-storage Location	*Operation*
A	PMM
B	VES
C	NOP
D	VMC
E	TIC A

An example showing the use of PCI exit using HALT I/O is shown in Table 13. Assume that the application program is performing an iterative operation, and the application program gives APAM the entry point to a PCI subprogram. APAM causes the channel program shown in Table 13 to be initiated. The application program may either wait or continue processing.

Operation of the channel program, shown in Table 13, is as follows. The CCW at location A initiates an operation such as partial matrix multiply (PMM), which is command chained to a vector element-by-element sum (VES) operation. The VES computes the difference between the input and output vectors of the previous operation. The VES is command chained to the NO OPERATION (NOP) at location C. The CCW for NOP is command chained to a vector move/convert operation (VMC), which moves the output of the PMM operation to the location of the input vector. The VMC is command chained through a TRANSFER IN CHANNEL command (TIC) to the PMM at location A in main storage.

This channel program loop is iterated until a HALT I/O instruction is issued by the CPU. The CCW's at locations C and D have their PCI bits set, and the PCI subprogram is first entered when the CCW at C is fetched from main storage. A system mask

bit is set to prevent any other interrupts, and processing continues.

The CCW at D is then fetched. Because the PCI bit is set and and because 2938 interrupts are disabled, array processor operations are temporarily suspended. While operations are suspended, the result of the VES operation is checked to determine whether that result is within established limits. If within limits, the PCI subprogram terminates by requesting a HALT I/O, which APAM issues to the array processor. If not within limits, the PCI subprogram exits without requesting a HALT I/O from the CPU.

The 2938 is now reenabled, the PCI from the VMC is "fielded," and array processing operations continue. In parallel with this processing, the PCI subprogram ignores the interruptions and waits for the next interruption from the PCI in the CCW at C. Processing continues until the result is within the required limits. This channel program may be performed concurrently with other CPU operations.

Another facility offered by APAM is that of dynamic exits for the purpose of interruption analysis. With this feature, the user may specify the entry point of an interruption subprogram and a code for the types of interruptions that will cause the interruption-analysis subprogram to be entered. The user may specify any combination of the following as the interruption codes: normal-completion, fixed-point overflow, floating-point overflow, termination by a HALT I/O instruction, or termination due to a permanent equipment malfunction. For example, if the 99,999th operation fails, the application program is not aborted. It may receive control on a unit check from the 2938 and continue processing. The application program may also, of course, use the 99,998 successful operations and exit.

There are several other notable features of APAM. Users may change the interruption code and/or the subprogram that is to receive control any number of times during execution of the application program. Also, requests may be issued to APAM by a user's subprogram. By use of an output-restricted deque, APAM places such requests ahead of any requests currently in queue. Control returns to the original point of interruption when the subprogram processing is completed. From his main application program, the user may also issue priority requests to the array processor. This facility allows such requests to be placed ahead of all requests already in queue for the 2938.

While running under OS/360, APAM allows the 2938 to be shared between regions in a multiprogramming environment. APAM can operate under the MVT, MFT-I, and PCP versions of OS/360. The OS/360 versions of APAM also have a dynamic syntax checking facility that may be optionally invoked. When requested, APAM can perform error checking of user parameter lists prior to dynamically building the requested channel program. If an error is discovered in a calling sequence, the user is notified by an appropriate message prior to the termination of the application program.

Table 14 SYSTEM/360 Model 44 operation timing

Operation	*Y*	*X*	*U*	*Methods (time in seconds)* 1	2	3	*Iterations*
SVE	1	2000		67.68	6.160		1000
SVE	1	*1500		23.264	2.192	0.800	500
SVE	1	500		7.248	1.808		500
SMY	2000	2000	1	44.272	5.088	3.712	500
VMC	500	500		14.992	2.160		500
VIP	1	2000	2000	50.688	4.912		500
VIP	1	1500	1500	38.544	4.128	2.608	500
VIP	1	500	500	11.472	2.304		500
VEM	500	500	500	16.800	2.784		500
CVM	250	500	250	203.712			50
CVM	250	500	250		8.048		500
SSQ	1	100		2.064	1.472		500
Matrix multiply				104.88		1.856	1

Table 15 SYSTEM/360 Model 65 operation timing

Operation	*Y*	*X*	*U*	*Methods (time in seconds)* 1	2	3	4	*Iterations*
SVE	1	2000		29.50	4.14			1000
SVE	1	1500*		11.25	1.81	.80		500
SVE	1	500		3.43	1.18			500
SMY	2000	2000	1	22.88				500
VMC	500	500		6.95	1.61			500
VIP	1	2000	2000	25.22	3.34			500
VIP	1	1500	1500	20.05	2.67	1.73	1.75	500
VIP	500	500	500	5.86	1.52			500
VEM	500	500	500	9.64	1.81			500
CVM	250	500	250	117.43				50
CVM	250	500	250		7.62			500
SSQ	1	100		1.13	1.06			500
Matrix multiply				60.85	1.26			1

Table 16 SYSTEM/360 Model 75 operation timing

Operation	*Y*	*X*	*U*	*Methods (time in seconds)* 1	2	3	4	*Iterations*
SVE	1	2000						
SVE	1	*1500		7.75	1.63	0.80		500
SVE	1	500		2.28	1.03			500
SMY	2000	2000	1	17.53	3.36	2.38	2.39	500
VMC	500	500		4.66	1.43			500
VIP	1	2000	2000	16.84	3.25			500
VIP	1	1500	1500	13.36	2.54	1.76	1.78	500
VIP	1	500	500	3.99	1.34			500
VEM	500	500	500	6.67	1.67			500
CVM	250	500	250	80.12				50
CVM	250	500	250		7.46			500
SSQ	1	100		0.70	0.86			500
Matrix multiply				42.88		1.28		1

Timing methods and results

The same FORTRAN programs were run on the three different models of SYSTEM/360 that support the 2938 array processor. Timing results of these operations are shown in Tables 14 to 16. Table 14 contains the results obtained when the 2938 is attached to a SYSTEM/360 Model 44, Table 15 contains comparable results for a Model 65, and Table 16 has the results for a Model 75.

The operations that were timed are shown in Table 17. Note that the first parameter in the parameter list that is passed to APAM (shown in the right column of the table) is a four-character mnemonic representation of the desired operation—shown by three alphabetic characters plus an asterisk. Matrix multiply was performed by command chaining partial matrix multiplies (PMM*). In all cases, the output array (Y) was produced in floating-point format. The input arrays (X,U) were in floating-point format except for the SVE* run (noted by the asterisk in the tables) which contains 1500 fixed-point halfword input X elements. The number of times a particular operation is performed is indicated in the iteration columns of Tables 14 through 16.

We now discuss the four methods of obtaining SMY* timing data. By the first method, the SMY* operation is performed completely in the CPU, using FORTRAN IV without the array processor. The subroutine used to perform a scalar multiply of a 2000 element X array 500 times by a floating-point U scalar is shown in Table 18. The resultant data is placed into the Y array. APTIM is a subprogram that prints the elapsed time between calls to APTIM on the FORTRAN output data set.

four timing methods

Table 18 FORTRAN IV program for scalar multiply in the CPU

```
Call APTIM
Do 2 I = 1,500
Do 2 K = 1,2000
2 Y (K) = X (K)*U
Call APTIM
```

Using the second method of collecting SMY* timing data, APAM constructs the channel program and initiates a number of array-processor operations equal to the iteration count. The program in Table 19 is an example of a build-execute-wait loop for a scalar multiply.

The following parameter list (from Table 19) is the long form of the parameter list that APAM is to build for execution. APAM

Table 17 Array processor operations timed

Sum of Vector Elements	SVE*	Vector Element Multiply	VEM*
Scalar Multiply	SMY*	Convolving Multiply	CVM*
Vector Move/Convert	VMC*	Sum of Squares	SSQ*
Vector Inner Product	VIP*	Matrix Multiply	

Table 19 FORTRAN IV program for scalar multiply using the array processor

```
Call APTIM
Do 3 I = 1,500
3 Call APAM ('SMY*,' 1, Y, 2000, 4, 0, X, 2000, 4, 0, U)
Call APTIM
```

does not return control until the 2938 has completed the operation. The parameters passed to APAM are interpreted as follows:

SMY*	Scalar multiply operation code
1	Execute in FIFO order and wait for completion
Y	Location of the resultant Y data array
2000	Number of elements in the resultant array
4	Four bytes between consecutive elements in the output array
0	Resultant array has floating-point format
X	Location of the input X array
2000	Number of elements in the input X array
4	Four bytes between consecutive X elements
0	X array is in floating-point format
U	Location of the input U scalar array in floating-point format by a default condition

Table 20 FORTRAN IV program for scalar multiply with the array processor using prebuilt CCW's

```
Call APTIM
Call APAM('EXC*',1, CCWS)
Call APTIM
```

The third method of timing the operation is to execute a prebuilt chain of CCW's as shown by the FORTRAN channel program in Table 20. The channel program is to be executed and control is not to be returned until the 2938 completes the operation. The APAM parameter list for the third method is interpreted as follows:

EXC*	Specification of a prebuilt channel program
1	Execute in FIFO order and wait for completion
CCWS	Location of the channel program to be executed

A fourth way in which the timing of the operation was measured makes use of a user PCI exit. The channel program consists of only two CCW's rather than 500. The first CCW contains a PCI and is command chained to a TIC. The second CCW (the TIC) transfers control back to the first CCW. Each time the PCI is accepted by the CPU, the user PCI exit is taken. When the desired number PCI's are received, a HALT I/O instruction is issued to the 2938. The FORTRAN program for the fourth method is the same as is shown for the third method in Table 20. The difference between the third and fourth methods is that the CCW's point to different channel programs.

timing results

Inspection of the data in Tables 14 to 16 shows that performance improvement is a function of the number of calculations performed. The larger the arrays and the more complex the operation, the greater the improvement. Also, as expected, the percentage increase in performance when the 2938 is attached to a SYSTEM/360 Model 44 is greater than the percentage increase in performance when the 2938 is attached to a more powerful system. The maximum improvement obtained was on the CVM operation in the SYSTEM/360 Model 44. Timing shows that the array processor path is 250 times as fast as the FORTRAN loop in the CPU.

In addition to the improvement in elapsed time, the CPU is available to do other work. In the case of the Model 44, the CPU can perform additional processing of the application program or

can handle other interrupts. When operating in an OS/360 multiprogramming environment, the CPU can process other application programs. It may still be profitable to perform an operation on the 2938 even if that operation could be done in the same or shorter time by the CPU alone, because this frees the CPU to do other work.

Table 21 summarizes timing results that were obtained for the matrix-multiply operation on a SYSTEM/360 Model 65 using FORTRAN IV with the H-level compiler and CPU processing (Column 2), together with timing results for APAM and an array processor (Column 3). The table shows system-performance-improvement ratios (Column 4) of FORTRAN IV with APAM and an array processor to FORTRAN IV with CPU processing for matrices of varying numbers of elements (Column 1). Timings of FORTRAN IV performance in seconds, with and without APAM, are shown in Columns 2 and 3. The FORTRAN IV matrix-multiply program is given in Table 22.

Performing a full matrix multiply on the 2938 requires repetitive partial-matrix multiplies. Each execution of a PMM produces one row in the resultant matrix. Therefore, to multiply two

Table 21 SYSTEM/360 model 65 performance comparison for matrix multiplication

Matrix size	FORTRAN IV *processing (seconds)*	*Array processor (seconds)*	*Performance improvement ratio*
100	60.85	1.26	48
92	47.56	0.94	50
84	36.22	0.74	48
76	26.84	0.57	47
68	19.26	0.41	46
60	13.23	0.28	47
52	8.62	0.19	45
44	5.24	0.12	43
36	2.86	0.07	40

Table 22 FORTRAN IV matrix-multiply program

```
      Call APTIM
      Do 100 I = 1, N
      Do 200 J = 1, N
      A(I, J) = 0
      Do 300 K = 1, N
      A(I, J) = B(I, K)*C(K, J) + A(I, J)
300   Continue
200   Continue
100   Continue
      Call APTIM
```

50 by 50 matrices, it is necessary to do 50 PMM's. The column of APAM-timing results was obtained by using the EXCP interface with APAM. A chain of PMM CCW's is executed. In the case of an 80 by 80 matrix, the list consists of 80 CCW's command chained together. Thus, with one SIO the full matrix multiply is accomplished.

The last column gives a comparison of the FORTRAN IV-CPU time versus the APAM-array-processor times. Performance improvement ratios are shown for the range of matrix-size values measured. As a rule, the performance improvement increases as the size of the matrices increases. Two exceptions may be noted in Table 21. The improvement factor decreases between matrix multiplies involving 92 and 100 square matrices. Another decrease in the performance ratio is shown when going from two 60-by-60-element matrices to two 68-by-68-element matrices. This is justified when one remembers the interface between the IOF and ARITH sections of the 2938 are two 32 element buffers. The optimum performance is obtained in a PMM when the size of the matrix is some multiple of 32. Any variation will cause the 32-element buffers to be partially used at certain times, thus slightly reducing the effectiveness of the 2938. As anticipated, an operation such as matrix multiply is best performed on the array processor in most cases.

Concluding remarks

The IBM 2938 Array Processor together with its access method (APAM) increases the processing power of SYSTEM/360 Models 44, 65, and 75 for operations on vectors and matrices. The auxiliary processing capabilities discussed have proved especially useful in such areas as seismic exploration, vibration analysis, turbulence research, and image enhancement. Because of the modular design of the present access method, it is possible to write signal processing subroutines that can perform certain common operations useful in these applications.

As another extension, specialized instructions can be incorporated into the array processor. One such instruction performs the fast Fourier transform,[10] which is an algorithm for computing the discrete Fourier transform for arrays of complex data. A specialized instruction is available for the scanning of arrays for their maximum data elements. Another special instruction produces the solution of difference equations of up to the fourth order by a single pass of the data. Higher-order or multiple-root equations may be solved by repeated passes of the data.

The advantage of implementing special-purpose operations by means of special equipment rather than with the standard instruction set is clearly shown by the timing results. Furthermore, the additional CPU availability for concurrent operations makes it profitable to have the 2938 perform operations that could be performed within the same time by the CPU. In a multiprogramming

environment, remote special-purpose processing becomes even more advantageous, especially if the remote processor can be shared by more than one application program. Such an environment exists when running OS/360 (MVT or MFT) with more than one program using the array processor.

CITED REFERENCES AND FOOTNOTES

1. A. D. Booth and K. H. V. Booth, *Automatic Digital Calculators*, Butterworths, Washington (1953).
2. A convolution of two vectors is a series of multiplications of corresponding elements of the vectors, followed by the summation of these products to produce one element of a resultant vector. The next resultant vector element is produced by shifting the two vectors by one element (relative to each other) and repeating the procedure. Mathematically, a convolution and a correlation are similar operations except that in a convolution the elements of one vector are taken in reverse order.
3. For operations that have relatively short execution times other methods have been used. One approach for this type of operation has been to add instructions to the vocabulary of a CPU. Whenever one of the new operations is to be performed, the corresponding new instructions are issued. Upon completion of the operation, the CPU proceeds to the next instruction in the usual manner. As execution time increases, this approach becomes less advantageous.
4. *Array Processor Access Method for IBM 2938 Model 2 with IBM SYSTEM/360 Model 65 or IBM SYSTEM/360 Model 75.* This Type III program and documentation, 360D 03.4.020, can be obtained through IBM Branch Offices. *Array Processes Access Method for IBM SYSTEM/360 Model 44,* may be similarly obtained by ordering 360D 03.4.019.
5. A. Padegs, "The structure of SYSTEM/360, Part IV-Channel design considerations, *IBM Systems Journal* **3**, Nos. 2 and 3, 165–180 (1964).
6. G. A. Blaauw and F. P. Brooks, Jr., "The structure of SYSTEM/360, Part I—Outline of the logical structure," *IBM Systems Journal* **3**, Nos. 2 and 3, 119–135 (1964).
7. D. E. Knuth, *The Art of Computer Programming, Volume I, Fundamental Algorithms*, 234–235, Addison-Wesley Publishing Company, Reading, Massachusetts (1968).
8. *IBM SYSTEM/360 Custom Equipment Description: 2938 Array Processor.* This publication, A24-3519, can be obtained through IBM Branch Offices.
9. R. A. Sebastian and T. J. Horrigan, "Why discriminate against the FORTRAN programmers?" *Software Age* **2**, No. 3, 8–12 (April 1968).
10. J. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Mathematics of Computation* **19**, No. 90, 297–301 (April 1965).