The Use of Array Processors for Numerical Modelling of Tidal Estuary Dynamics

by D. Prandle*, E.R. Funke**, N.L. Crookshank*** and R. Renner****

1.0 INTRODUCTION

The use of array processors for the numerical modelling of estuarine systems is discussed here in the context of "hybrid modelling", however, it is shown that array processors may be used to advantage in independent numerical simulations. Hybrid modelling of tidal estuaries was first introduced by Holz (1977) and later by Funke and Crookshank (1978). In a hybrid model, tidal propagation in an estuary is simulated by dynamically linking an hydraulic (or physical) scale model of part of the estuary to a numerical model of the remaining part in a manner such that a free interchange of flow occurs at the interface(s). Typically, the elevation of the water surface at the boundary of the scale model is measured and transmitted to the numerical model. In return, the flow computed at the boundary of the numerical model is fed directly into the scale model.

This approach enables the extent of the scale model to be limited to the area of immediate interest (or to that area where flow conditions are such that they can be most accurately simulated by a scale model). In addition, since the region simulated by the numerical model can be extended almost indefinitely, the problems of spurious reflections from downstream boundaries can be eliminated.

In normal use, numerical models are evaluated on the basis of computing requirements, cost and accuracy. The computer time required to simulate one tide cycle is, in itself, seldom of interest except in so far as it affects the above criteria. However in hybrid modelling this parameter is often paramount since concurrent operation of the numerical and scale models requires that the former must keep pace with the latter.

The earlier hybrid model of the St. Lawrence (Funke and Crookshank, 1978) involved a one-dimensional numerical model of the upstream regions of the river. However, future applications are likely to involve extensive two-dimensional numerical simulation. Consequently the

----

* Visiting Scientist, **Senior Research Officer, ***Systems Manager, Hydraulics Laboratory, National Research Council of Canada, Ottawa, Canada.
**** Chief Scientist, Canadian Astronautics Ltd., Ottawa, Canada.

computational power required will be considerable and almost certainly in excess of the power of present-day mini-computers. The use of a substantial main frame machine is complicated by both the cost and the requirement for a high priority real time service. Thus the use of an array processor/mini-computer combination was examined to bridge this gap in computational power.

Section 2 describes a hybrid model of the Bay of Fundy with the details of the numerical scheme given in section 3. Sections 4, 5 and 6 describe, in some detail, various aspects relating to the usage of the array processor. Finally section 7 provides a comparison of the performance of the array processor against that of both a mini-computer and two main frame machines.

2.0 THE BAY OF FUNDY HYBRID MODEL

In order to subject the development project to the rigors of a practical application, a suitable estuary was selected. The Bay of Fundy offered several advantages. For one, there exists a proven explicit model (Greenberg, 1976) which covers both the Bay of Fundy and the Gulf of Maine, down to the Continental Shelf. Secondly, there is a considerable interest in the electric power potential of the Fundy tide and it was felt that the experience gained from the pilot model study may benefit future investigations whenever the Bay of Fundy tidal power development progresses to the engineering design stage. With these considerations in mind, the construction of a pilot hybrid model was initiated with Cumberland Basin and Shepody Bay forming the physical model and the remainder of the estuary down to the Continental Shelf forming the numerical model. Fig. 1 illustrates the general outline of this estuary, and includes the schematization employed for the finite difference scheme. The small "boxed-in" area in the upper right hand corner of the diagram is simulated by a physical model. Fig. 2 illustrates the outline of this scaled model and its relationship to the numerical model. The computer in this hybrid model serves the dual functions of being both host to the array processor and at the same time, being the data acquisition system and feedback controller for the discharge control pump (Funke, Crookshank and Wingham, 1980).

Fig. 3 gives the timing diagram of a typical hybrid model. It should be noted that the "hatched" pulses represent the regular clock pulses which, for the Bay of Fundy model, occur every 0.3 seconds. Shortly after each pulse, a data transfer takes place which transmits the last calculated discharge value Q_I from the array processor to the control computer. The data acquisition phase follows immediately, monitoring among other variables, the water elevation H_I which is then transmitted back to the array

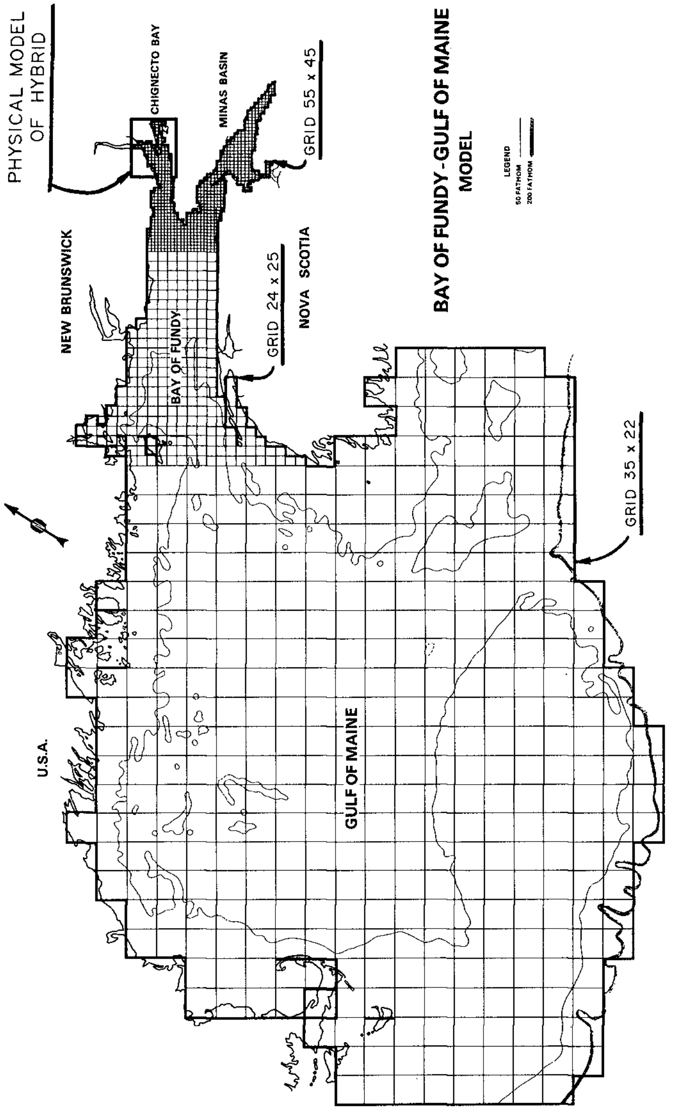


FIG. 1

SCHEMATIC REPRESENTATION OF BAY OF FUNDY

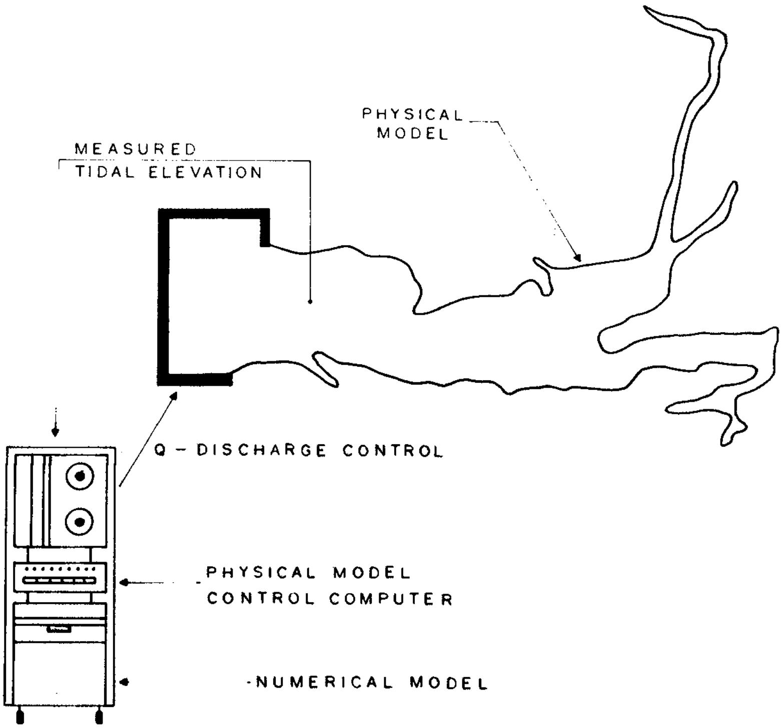


FIG. 2 HYBRID MODEL OF CUMBERLAND BASIN AND SHEPODY BAY

processor. Next comes the model control phase and while the control computer is engaged in achieving the required discharge, the array processor must solve one time step of the numerical model. Since clock pulses for the Bay of Fundy hybrid model occur every 300 milliseconds, the computational time available for one time step of the numerical model must be somewhat smaller. In fact, sufficient time must remain to complete all the data acquisition and the data transfer between the host computer and the array processor. In addition one wants to have some margin of safety so that the "ping-pong" interchange between the two machines can never get out of synchronism.

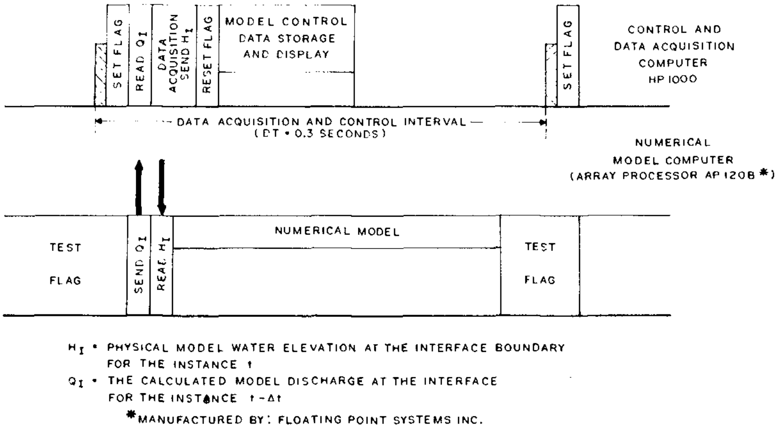


FIG. 3 TIMING DIAGRAM FOR HYBRID MODEL

3.0 THE EXTENT OF THE MATHEMATICAL MODEL

The estuary dynamics of the mathematical model for the present Bay of Fundy is based on the following equations:

x-motion: $$\frac{\partial U}{\partial t} + g\frac{\partial Z}{\partial x} + f\,\frac{U\cdot(U^2 + V^2)^{1/2}}{(D + Z)} - \Omega\cdot V = 0$$

y-motion: $$\frac{\partial V}{\partial t} + g\frac{\partial Z}{\partial y} + f\,\frac{V\cdot(U^2 + V^2)^{1/2}}{(D + Z)} + \Omega\cdot U = 0$$

continuity: $$\frac{\partial Z}{\partial t} + \frac{\partial}{\partial x}\,[U\cdot(D + Z)] + \frac{\partial}{\partial y}\,[V\cdot(D + Z)] = 0$$

where U and V are the velocities in the x and y direction,
D is the depth below mean water,
Z is the tidal elevation about the mean,
f is the friction term, and
Ω is the Coriolis parameter.

Fig. 4 shows the schematization using a finite difference, explicit method with a staggered mesh and forward difference

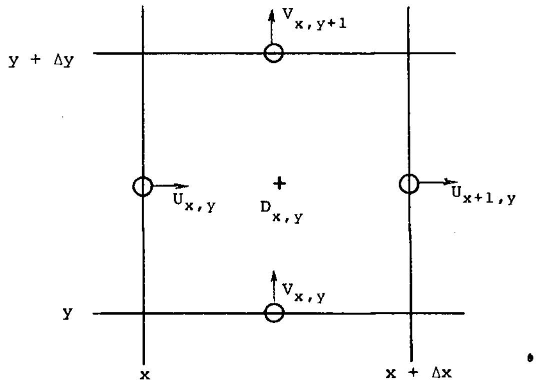


FIG. 4 DEFINITION OF FINITE GRID ELEMENT

in time. The differential equations are thus approximated by the following.

$$U'_{x,y} = (U_{x,y}/\Delta t + g(Z_{x,y} - Z_{x-1,y})/\Delta x - \Omega\cdot\overline{V})/(1/\Delta t + F_x)$$

where:

$$\overline{V} = (V_{x,y} + V_{x-1,y} + V_{x,y+1} + V_{x-1,y+1})/4$$

and

$$F_x = f\cdot(U_{x,y}{}^2 + \overline{V}^2)^{1/2}/((D_{x,y} + D_{x-1,y} + Z_{x-1,y})/2)$$

$$V'_{x,y} = (V_{x,y}/\Delta t + g(Z_{x,y} - Z_{x,y-1})/\Delta y + \Omega\cdot\overline{U})/(1/\Delta t + F_y)$$

where:

$$\overline{U} = (U_{x,y} + U_{x+1,y} + U_{x,y-1} + U_{x+1,y-1})/4$$

and

$$F_y = f\cdot(\overline{U}^2 + V_{x,y}{}^2)^{1/2}/((D_{x,y} + D_{x,y-1} + Z_{x,y} + Z_{x,y-1})/2)$$

and

$$Z'_{x,y} = \Big\{Z_{x,y}/\Delta t + [U_{x+1,y}\cdot(D_{x+1.y} + D_{x,y} + Z_{x,y} + Z_{x+1,y})$$
$$- U_{x,y}\cdot(D_{x,y} + D_{x-1,y} + Z_{x,y} + Z_{x-1,y})]/(2\cdot\Delta x)$$
$$+ [V_{x,y+1}\cdot(D_{x,y+1} + D_{x,y} + Z_{x,y+1} + Z_{x,y})$$
$$- V_{x,y}\cdot(D_{x,y} + D_{x,y-1} + Z_{x,y})]/(2\cdot\Delta y)\Big\}$$

Fig. 1 shows that the area to be modelled is subdivided into three zones. The Gulf of Maine is represented by a 35 x 22 grid. The Bay of Fundy area has three times the resolution with a 24 x 25 grid. Finally the upper reaches of the Bay have again three times the resolution with a 55 x 45 grid.

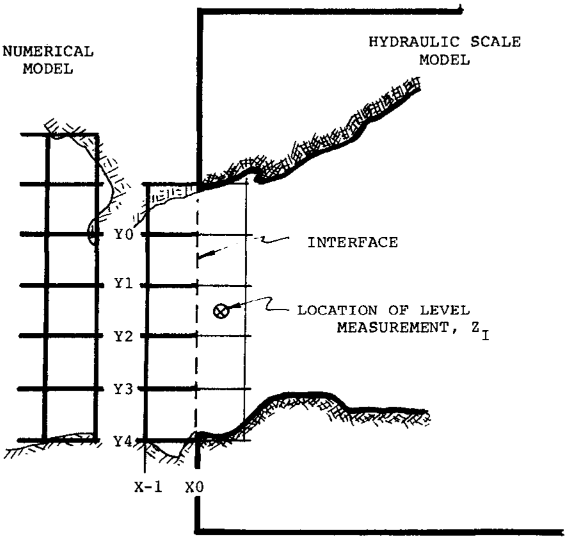
Fig. 5 shows that five finite grid elements form the boundary with the scaled model. It is anticipated that each of these five elements could control an appropriate interface pump. However, for the pilot model now under construction, only one elevation, Z_I, is measured and the numerical model supplies the average discharge through the boundary by means of the averaging formula given in Fig. 5.

4.0 A BRIEF DESCRIPTION OF ARRAY PROCESSOR HARDWARE

An array processor (AP) is a digital data processor which is specifically and optimally designed to process long data vectors. Typically, one can say that the longer the data vector or vectors, the more advantageous is the array processor as a "number cruncher".

Array processors are peripheral to so-called host computers. Their advantage must, of course, be measured relative to their host computer. For this reason, it is quite common now to find array processors interfaced to mini-computers rather than to larger main frame machines. The list of references suggests some papers which offer more technical information on various array processors.

Fig. 6 shows the major components of a typical array processor "Floating Point Systems Inc. AP120B", i.e. the machine used for this particular study. From this diagram several pertinent features can be recognized:



ASSUME: $Z_{X0,Y0} = Z_{X0,Y1} = Z_{X0,Y2} = Z_{X0,Y3} = Z_{X0,Y4} = Z_I$

$$Q_I = \sum_{i=Y0}^{Y4} \Delta y \cdot U_{XO,i} \cdot (D_{XO,i} + D_{X-1,i} + Z_{XO,i} + Z_{X-1,i})/2$$

FIG. 5 INTERFACE DETAILS BETWEEN NUMERICAL AND SCALED MODEL

(a) The AP has separate and independent memory components for data, program store and table constants. This permits not only some parallel processing but also an optimum choice of word length for instructions and data respectively. For example, the AP120B has a 64 bit instruction word which may control up to ten different operations more or less at the same time. On the other hand, the data word is 38 bits long with 28 bits used as mantissa and 10 bits as exponent. This is a worthwhile improvement over the usual 32 bit data formats especially for the type of problem described by this paper.

(b) The AP has parallel arithmetic processors which may operate on data concurrently.

(c) Data processing may take place in a "pipeline" fashion so that data words move progressively through successive stages. Each stage may require in the order of

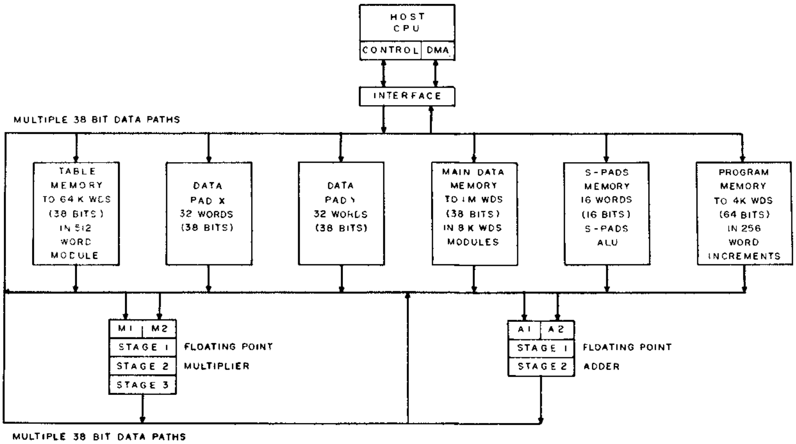


FIG. 6 GENERAL SYSTEM DIAGRAM FOR AP122B ARRAY PROCESSOR

167 nanoseconds. Once the "pipeline" is filled, solutions are returned back to the data memory at the same 167 nanosecond rate. It is both the parallel processing and the "pipelining" which gives these processors their phenomenal speed.

(d) The interface between the host computer and the AP is of particular importance in appreciating the operation of the machine in relation to its host. In the usual configuration depicted by Fig. 6, all AP-programs and data come from the host and results are returned to the host. An executive program in the host keeps track of the programs which are required in the AP program store and if a particular program which is being called is not, at that time, resident in the AP, it must be transferred there into whatever free area is available. If the program store is filled to capacity, then the last program in will be the first program to be overlaid, and hence destroyed.

Data transfer to and from the AP is usually costly in time. Consequently an awareness of these operational and hardware features can affect the manner in which an AP program is written for best performance.

5.0 ARRAY PROCESSOR APPLICATION TO BAY OF FUNDY MODEL

Fig. 7 illustrates the particular computer and

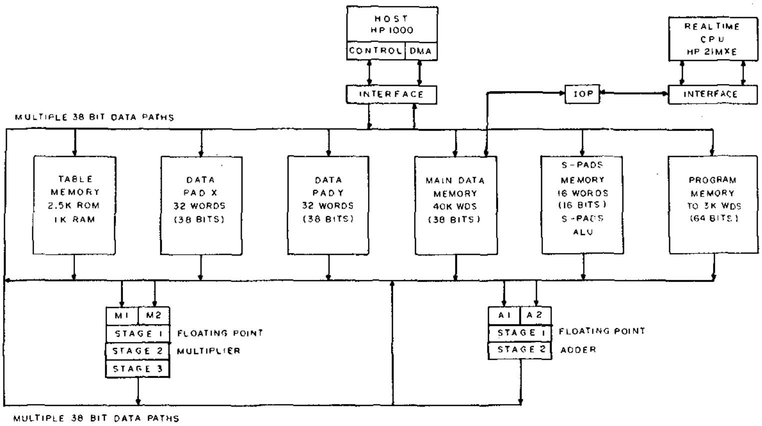


FIG. 7 THE ARRAY PROCESSOR CONFIGURATION AS FOR B OF F MODEL

array processor configuration which is being used for the Bay of Fundy pilot hybrid model. It may be noted that the host computer is a Hewlett-Packard HP-1000 model 45, while the data acquisition and on-line, digital control computer is the HP-21MXE computer. Although the configuration of Fig. 6 could have served the requirements of the hybrid model, the configuration of Fig. 7 was selected in order to get a better overall system utilization. This is of particular importance since the hybrid model is not the only real time activity which is being supported concurrently by this H.P. computer system (Funke, Crookshank and Wingham, 1980).

The DMA interface to the host computer in Fig. 7 is the usual channel for transmission of AP-programs and for initial model constants. The entire Bay of Fundy numerical model, as described in section 3.0, is down-loaded in this way prior to commencement of actual model operation. Once in operation, data related to the tidal elevation is transmitted from the data acquisition computer to the AP via the input/output processor box (IOP) and the resultant discharge data travels on the same channel in the opposite direction. In this manner, the array processor is an autonomous, dedicated numerical model, completely freeing the host computer for other activity.

A suitable "logical" switch was provided to run the numerical model of the Bay of Fundy either as a hybrid model or as a completely independent numerical model. In the former case a boundary exists at the entrance to the Cumberland and Shepody Basins and boundary information is transmitted via the IOP. In the latter case, this boundary does not exist as the two basins are included in the numerical model.

In order to monitor the progress of tidal propagation through the numerical model, the solutions for tidal elevation at each grid point are also transmitted to the data acquisition and control computer at each control step. In this manner one may treat the data in a similar fashion to other data which were acquired through instrumentation on the physical model.

6.0 PROGRAMMING ARRAY PROCESSORS

Whereas the array processors offer substantial improvements in processing speed, the effort required to exploit their power may still be substantial. For this reason it is a definite advantage to have the help of an expert consultant who can quickly solve the usual "teething" problems and who can pilot the project around the various pitfalls.* However, there is a significant and promising development in progress which may overcome many obstacles.

There are four different ways by which the Floating Point Systems Inc. AP120B may be programmed. Each of these offers certain advantages or disadvantages which must be traded off.

6.1 FORTRAN Calls to Existing Library Subprograms

Fig. 8 gives a typical example of a program task which requires various vector and matrix operations. The first portion in Fig. 8 describes this task as a conventional FORTRAN code. Following this, one may recognize calls to various subroutines which serve

(a) to initialize the AP,
(b) to transfer data from the host to the AP, and
(c) to cause a wait until the transfer of data is complete.

It is worth noting that the data memory in the AP is addressed here in terms of absolute addresses and these must be generated before the data transfer calls can

---

* Canadian Astronautics Ltd.
1024 Morrison Dr., Ottawa, K2H 8K7 Canada.

```
C******* POTENTIAL CALCULATION ***************************************
C
C=================================================================
C     ORIGINAL FORTRAN
C
C        SUBROUTINE EX2
C        COMMON /B/PHIB(100,10),HB(100,10),PKB(100),DS12
C        DO 1 J=1,9
C        DO 1 I=1,100
C 1      PHIS(I,J+1)=PHIB(I,J)+DS12*PKB(I)*(HB(I,J+1)+HB(I,J)
C        RETURN
C        END
C=================================================================
C
         SUBROUTINE EX2
         COMMON /B/PHIB(100,10),HB(100,10),PK(100),DS12
C------AP MEMORY LAYOUT
         IDS12=0
         IPKB=1
         IHB=IPKB+100
         IPHIB=IBH+1000
C------INITIALIZE THE AP
         CALL APINIT (0,0,STATUS)
         IF (STATUS.LT.0) CALL ERROR
C------PUT OUT THE DATA TO AP
         CALL APPUT(PHIB,IPHIB,1000,2)
         CALL APPUT(HB,IHB,1000,2)
         CALL APPUT(PKB,IPKB,100,2)
         CALL APPUT(DS12,IDS12,1,2)
         CALL APWD
C
C------DO THE COMPUTATION
C
C      AP COMPUTATION TIME IS 2.3 MS FOR 167 NS MEMORY, 3.7 MS FOR
C      333NS MEMORY, EXCLUSIVE OF HOST SYSTEM OVERHEAD
C
         CALL VSMUL(IPKB,1,IDS12,IPKB,1,100)
         CALL VADD(IHB+100,1,IHB,1,IHB,1,900)
         JHB=IHB
         DO 1 J=1,9
         CALL VMUL(IPKB,1,JHB,1,JHB,1,100)
  1      JHB=JHB+100
         CALL VADD(IPHIB,1,IHB,1,IPHIB+100,1,900)
         CALL APWR
C------GET THE RESULTS FROM AP
         CALL APGET(PHIB(1,2),IPHIB+100,900,2)
         CALL APWD
         APRLSE
         RETURN
         END
```

FIG. 8   EXAMPLE AP PROGRAM FOR CALLS ON AP-MATHEMATICAL LIBRARY be made. Other arguments specify typically how many elements are to be transferred and what format conversion is to take place.

The subsequent calls deal with the actual solution of the problem. One may recognize vector multiply and vector addition operation which make reference to the various arrays in terms of their addresses in AP memory. In order to make these routines as general as possible, they have been designed to permit operation either on consecutive elements (i.e. arguments No. 2, No. 4 and No. 6 are set to 1) or on alternate or arbitrarily spaced arguments. It is typical for the list of arguments to be organized as "SOURCE1", "SOURCE2" and "DESTINATION". For each of these the order is always "WHERE", "HOW MANY" and "NUMBER OF SKIPS - 1". Prior to the data transfer from AP to host a "WAIT FOR AP READY" must also be invoked.

It should be noted that the DO-loop and the calculation of the "JHB" parameter are executed in the host computer and for each pass through the DO-loop a transfer of subroutine arguments to the AP will be implemented. This is not the fastest way of running the solution but it does offer a relative simplicity in implementation.

The disadvantages of this approach are:

(a) Programming is limited to existing algorithms in the various libraries supplied by the manufacturer,
(b) Special requirements, such as conditional branches, require FORTRAN coding in the host computer with the consequent loss of speed due to repeated interchange of information between the host and the AP,
(c) Addressing of variables and arrays in the AP must be implemented in terms of absolute addresses with a subsequent loss of the convenience and power of a mnemonic address structure which is inherent to FORTRAN and
(d) Although each AP subprogram has been coded in an optimum fashion, any special requirements, which could benefit from some of the various hardware features of the AP, cannot be accommodated.

6.2 Programming with the Vector Function Chainer Language

The Vector Function Chainer is an AP programming language of a somewhat higher power than the simple calling of precoded library subprograms. This language allows not only the creation of new AP library subprograms, but it also permits some simple, FORTRAN-like statements for execution in the array processor rather than the host computer.

Fig. 9 gives an example of a subprogram for a

```
"******* MVADD = MATRIX/VECTOR ADD **********************************
"
"       DEFINE MVADD(A,I,B,J,C,K,NRC,NCC)
"
"       ADD VECTOR B TO EVERY ROW OF MATRIX A, PUTTING THE RESULT IN C
"
"       A - ADDRESS OF MATRIX A
"       I - INCREMENT BETWEEN ELEMENTS OF A
"       B - ADDRESS OF VECTOR B
"       J - INCREMENT BETWEEN ELEMENTS OF B
"       C - ADDRESS OF DESTINATION MATRIX C
"       K - INCREMENT BETWEEN ELEMENTS OF C
"       NRC - NUMBER OF ROWS IN C (AND A)
"       NCC - NUMBER OF COLUMNS IN C (AND A)

"THE MATRICES ARE STORED IN COLUMN ORDER. THUS I AND K ARE INCREMENTS
"BETWEEN ELEMENTS IN A COLUMN. WE MUST COMPUTE THE INCREMENT BETWEEN
"ELEMENTS IN A ROW.
"
"       LOCAL AR,CR
"
        AR = I * NRC                "COMPUTE 'A' ROW INCREMENT
        CR = K * NRC                "COMPUTE 'C' ROW INCREMENT
"
LOOP:   CALL VADD(A,AR,B,J,C,CR,NCC)        "ADD TO A ROW
        A = A + I                           "ADVANCE 'A' POINTER
        C = C + K                           "ADVANCE 'C' POINTER
        NRC = NRC - 1                       "DECREMENT ROW COUNTER
        IF NRC < 0 GOTO LOOP                "GO BACK IF NOT DONE
        END
"
"
```

FIG. 9 EXAMPLE AP PROGRAM USING VECTOR FUNCTION CHAINER

matrix/vector addition which was created using the vector function chainer. This example illustrates some of the features of this language such as the calling of other existing subprograms, the creation of absolute addresses by arithmetic statements and the use of the logical IF-statement.

Fig. 10 shows the procedure by which a vector function chainer program is implemented. The source code of the program is first processed by the vector function chainer and the resultant output is a "second stage" source code in the AP assembler language. This assembler must also process the code which is then fed through the AP linking loader which serves to satisfy calls to the AP-library. The result of these operations leads to a third-stage source code which is in the host FORTRAN language. However, this code is quite unreadable as it consists of no more than a subroutine definition and termination statements and a long list of DATA statements with integer values. Each integer word is a quarter of a 64 bit instruction word which forms part of the desired AP program.

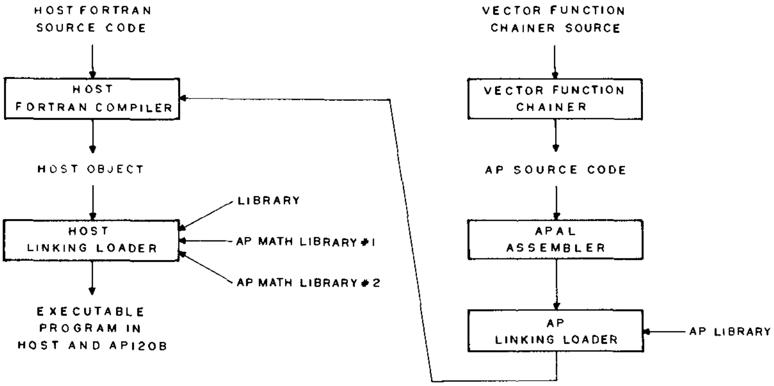


FIG. 10 USE OF VECTOR FUNCTION CHAINER

This third stage source program now represents the newly created member of an AP subprogram library. Before execution it must be compiled together with its calling FORTRAN program by the host FORTRAN compiler and then loaded in the usual fashion.

The vector function chainer offers greater programming power than the approach described under section 3.1. It is equally cumbersome in the management of absolute addresses but since addresses and additional branches defined by the vector function chainer language are computed within the AP, the repeated information transfer between the host computer and the AP is eliminated and a considerable speed-up of the solution times is achievable. The explict model of the Bay of Fundy was coded in this manner.

6.3 Programming by Using the AP-FORTRAN Compiler

A more recent addition to the bag of tricks for programming is a FORTRAN compiler for the array processor. It offers potentially many significant advantages over any other approach and promises to make the array processor a truly general purpose computer which can bring low cost, high speed computation into the reach of anyone who has a need for it.

In order to code a task in AP FORTRAN, it is necessary to define that portion of a program which is to run on the array procesor as distinct from the host computer. This portion must be coded as a standard FORTRAN subprogram. If all of the task is to run on the AP, it is still necessary to have a host program that simply states:

```
READ ARG1,ARG2
CALL NAME(ARG1,ARG2, ... ARGN)
WRITE ARGN
END
```

The subroutine NAME must be processed by the AP FORTRAN compiler which will take care of all data transfers to and from the AP and the associated wait-calls. It can handle any type of linear or non-linear functional relationships and any multi-dimensional array configuration. However, the usual data input/output function via the host computer's peripherals must be looked after by the host computer.

Fig. 11 shows the procedure for implementing an AP pro-

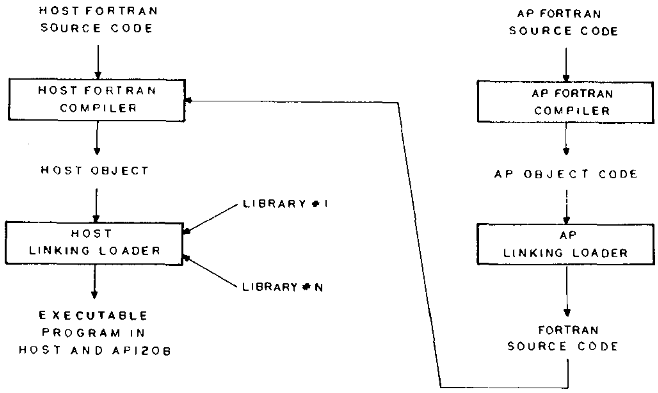


FIG. 11 USE OF AP FORTRAN COMPILER

gram by the AP FORTRAN compiler. The source code of the subroutine 'NAME' is first processed by the AP FORTRAN compiler. This must be done on a larger 32 bit computer as the compiler is not, at present, operational on 16 bit computers. However, its output may be run on 16 bit host computers which provide, after linking-loading, a secondary source code in host FORTRAN. As before, this source code consists substantially of DATA statements only and looks quite similar to the third stage source code produced by the vector function chainer described in section 6.3.

At this time, the community of users of an AP-FORTRAN compiler is still relatively small and general experience in its use must yet be established. However, for the purpose of the Bay of Fundy hybrid model study, an investigation was carried out for the purpose of:

(a) establishing the suitability of either the explicit or the implicit method for numerical models with regard to array processor operations, and
(b) establishing the effectiveness of the AP-FORTRAN compiler vis à vis the Vector Function Chainer as a means for implementing the implicit model on the AP.

Table 1
COMPARISON OF AP-FORTRAN WITH VECTOR FUNCTION CHAINER

	AP PROGRAM STORE RQRD	PROCESSING SPEED PER TIME STEP**	NO. OF DAYS RQRD TO IMPLEMENT PROGRAM*
VECTOR FUNCTION CHAINER	2700	1.5 s	7 days
AP-FORTRAN COMPILER	2200	1.5 s	1 day

* From an original statement of the model in terms of a correctly working program in host computer FORTRAN.
** 48 steps per tidal cycle

The comparison of Table 1 favours the AP-FORTRAN compiler. This came as a surprise because other users had indicated that both the required program store and the solution time would increase as a consequence of using the AP-FORTRAN compiler.* It may be possible that the nature of the particular program or the manner in which the program was coded using the vector function chainer could both affect the results. Nevertheless the authors' experience has been most encouraging even though there still are some minor errors in the AP-FORTRAN compiler.

6.4 Programming by Using the AP Assembler Language

In order to get the greatest processing speed with the least amount of required program store, it is necessary to use assembly language. This option was considered as a last resort for the Bay of Fundy hybrid model if other programming methods had not brought the solution speed within the real time constraints imposed by the physical model.

---

* Verbal communications

The assembly language permits direct control over all registers, data buses and arithmetic units. The price for this additional flexibility and power is the greater language complexity. As a result it is difficult and costly to learn the language and very time consuming to create an error-free program. Coding by the AP assembly language is considered practical only for those situations where the additional speed justifies the additional cost in program development effort.

7.0 COMPARISON OF AP PERFORMANCE TO OTHER COMPUTERS

Comparisons between computers are meaningful only in terms of specific benchmark programs which contain a specific mix of computational operations. For this reason it is necessary to point out that this comparison applies strictly to the solution of a system of finite difference equations describing estuary dynamics.

The original numerical model of the Bay of Fundy (Greenberg, 1976) uses an explicit method, a schematization similar to Fig. 1 and a time step of 30 seconds in prototype time. Because of the particular interests in the Minas Basin, that area was originally schematized with a finer grid than the one shown in Fig. 1 However, the hybrid model of the Cumberland and Shepody Basins does not require this detailed representation of the Minas Basin and therefore the schematization of Fig. 1 could be adopted. As a consequence, the time step could also be increased to 1 minute in prototype time. The execution times which are listed in Table 2 apply to one semi-diurnal cycle of a tide and, the execution times for the explicit model are based on a time step of 1 minute.

Since the original Greenberg model was not run under exactly these conditions, the time for the CDC CYBER 74 (equivalent to a CDC 6600) can only be estimated.

A finite difference implicit model of the Bay of Fundy was tested with 15 minute time step in prototype time. For the particular requirements of the hybrid model, this implicit method is not considered economically justified. At $1.00 per word for AP data storage, this model is substantially more expensive to implement.

In assessing the results shown in Table 2, one additional factor is the usefulness of an in-house machine for other applications. In buying time on a main-frame machine, the cost involved includes a significant proportion relating to peripheral equipment which may not be required for present purposes. Thus while it is difficult to obtain accurate costs for main frame time, Table 3 shows that the complete array processor/mini-computer package costs less than $150 000 - a figure considerably less than the annual expenditure involved with many large model studies.

Table 2
Comparison of Execution Times for Fundy Models

	EXPLICIT MODEL 1 MINUTE TIME STEP 750 STEPS/CYCLE		IMPLICIT MODEL 15 MINUTE TIME STEP 48 STEPS/CYCLE	
	MINUTES	MEMORY	MINUTES	MEMORY
CDC CYBER 74	2 to 2.5			
IBM 3032	5			
HP1000, MOD. 45	75.	60K	128	145K WDS*
AP-120B	2.5	40K WDS**	1.2	40K WDS

* A modification to the algorithm reduced this to 75K WDS
**Considerable memory savings could be achieved with additional programming effort.

Table 3
CAPITAL COST FOR MINI-COMPUTER/AP SYSTEM

(January 1980)

1.	HP1000, MODEL 45 - 128 K WORDS MEMORY - 20 MBYTE DISC, - 2648 VIDEO GRAPHICS TERMINAL	$45 000.
2.	AP-120B ARRAY PROCESSORS WITH - 3K PROGRAM STORE, - 1K TABLE RAM, - 40K DATA MEMORY (167 ns), - IOP	$81 130.
	- EXTENDED SOFTWARE	$ 8 475.
	- AP-FORTRAN COMPILER	$ 8 500.
	U.S.	$143 105.

8.0 GENERAL REFERENCES

-- Funke, E.R. and N.L. Crookshank (1978), "A Hybrid Model of the St. Lawrence River Estuary", Proc. 16th Coastal Engineering Conf., Hamburg.

-- Funke, E.R., N.L. Crookshank and M. Wingham (1980), "An Introduction to GEDAP - An Integrated Software System for Experiment Control, Data Acquisition and Data Analysis", Hydraulics Laboratory Technical Report LTR-HY-75, NRC, Ottawa, Canada.

-- Greenberg, D.A. (1976), "Mathematical Description of the Bay of Fundy-Gulf of Maine Numerical Model", Technical Note No. 16, Marine Environmental Data Service, Environment Canada, Ottawa.

-- Holz, K.P. (1977), "Hybrid Models, A Study on Their Principle and Realization", Proc. 7th Conf. IAHR, Vol. 6, pp. 674-678.

9.0 ARRAY PROCESSOR REFERENCES

-- Alexander, P. (1979), "Array Processors", Machine Design, pp. 87-92, August 23, 1979.

-- Caspe, R.A. (1978), "Array Processors", Mini-Micro Systems, pp. 54-64 (Comparison of Processors).

-- Hufnagel, S.P. (1979), "Comparison of Selected Array Processor Architectures", Computer Design, pp. 151-158, March.

-- Strelchun, J. (1979), "Array Processor Responds in Real Time", Electronics, pp. 118-124, August 16, 1979.

-- Wiley, P. (1979), "Interfacing Peripherals Directly to an Array Processor", Computer Design, pp. 158-164, August 1979.

-- Wittmayer, W.R. (1978), "Array Processor Provides High Throughput Rates", Computer Design, pp. 93-100.