Some Remarks on High-Speed Automatic Computers and their Use in Meteorology

By GEORGE W. PLATZMAN
University of Chicago

(Manuscript received 30 June 1952)

Abstract

The logical design of an electronic digital computer is described. A simple computation problem is proposed; this is used to illustrate the main steps leading to formulation of a problem for machine computation — the propram and the flow diagram. In conclusion, some general comments are made concerning areas of application of high-speed computers in meteorology.

High-speed digital computers now are to some extent available to meteorologists for research. Doubtless there will be intensive developments in certain fields of theoretical and applied meteorology where heretofore progress has been inhibited by the prodigous amounts of computation required to yield a useful result. As this work advances, it is likely to take on some of the aspects of an engineering science, and the underlying principles of the use of computers for meteorological problems may tend to become obscured by a superstructure of special methods of computation and procedure.

It is inevitable that high-speed computers will open new avenues of investigation in theoretical studies of the atmosphere, and although among meteorologists there is widespread latent interest in these prospects, it is probably too early to predict when to expect results of practical value — say, to the forecaster. Nevertheless, it seems rather evident that to maintain a well-balanced approach to problems that offer some promise of yielding to computation methods, there should be effective communication between those who gain experience in the techniques of computation, and those whose contacts are mainly with applied meteorological problems — for example, in the fields of forecasting or synoptic meteorology.

With this thought in mind, the writer has undertaken to sketch a broad view of some of the steps that enter into the task of domesticating a high-speed computer for meteorological purposes, and although the view is from a very limited level of experience in these matters, it may nevertheless help to illuminate for the meteorologist this novel area of research and to implement his interests in a field that he may sense is intriguing but which otherwise may seem obscure. The results of this endeavor are reported in what follows; they are grouped under four headings: (1) a computer, (2) a problem, (3) a computation, (4) concluding remarks.

The substance of this discussion has been taken from the reports by Goldstine and von Neumann, to which reference is made at the end of the paper. The reader should consult these reports for a more systematic and thorough treatment of the logical design of a high-speed computer and a more detailed exposition of methods of preparing problems for computation.

1. A computer

An electronic digital computer is an extremely complex instrument containing many thousands of components and involving application of the most advanced techniques of electronic engineering. Fortunately, it is possible to comprehend the logical structure of the machine without understanding the design and construction whereby this logical structure is achieved, just as it is possible to know the logical structure of an automobile without understanding the construction of the various components or even the manner in which they accomplish the end result. Indeed, this is not surprising because in point of fact, the logical structure of the computer must be formulated more or less independently of the specific engineering techniques by means of which its physical counterparts ultimately are realized.

The basic logical elements of an electronic digital computer are (i) the *arithmetic* organ, (ii) the *memory*, (iii) the *control*. In addition, there must be provision for transfering data into and out of the machine — namely, an *input-output* device. The significance of the primary logical elements can be illustrated by a simple example. Let a and b stand for two numbers which, at a certain stage in a particular computation, must be combined arithmetically to yield a derived number, $F(a, b)$; for example, $F(a, b) = a+b$, or $a-b$, or $a \times b$, etc. Initially, a and b are stored in the memory; the control arranges their transfer to the arithmetic organ, instructs this organ to produce $F(a, b)$, and returns this derived number to the memory (see figure 1). (Certain operations involve only a single number; for example, $F(a) = -a$, or $F(a) = |a|$, or $F(a)$ = the first four digits of a, and so on.)

The arithmetic organ contains a small number of *registers* — analogous to the "dial" registers of an ordinary desk calculator — each of which can hold one number (that is, a series of digits) at any instant. Not many registers are required if, as in most computers, the arithmetic organ performs only one operation at a time. When a number held in a register is no longer needed, the control can erase it, or send it to a specific location in the memory. If the specified memory location already holds a number, this number is erased and the number sent from the register takes its place.

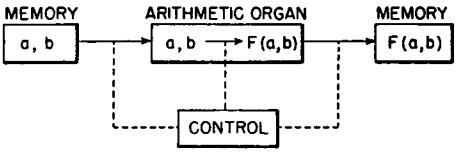

Figure 1. The basic elements and their functions.

The usefulness of a high-speed computer depends to a large extent on the *versatility of the control*, and the *capacity of the memory*. Every computation of sufficient scope to justify the use of a high-speed computer will involve a great many intermediate numbers that do not represent the final result but are produced at some stage of the computation and set aside to be used at a later stage. Furthermore, in some problems the initial data as well as the final computed result may in themselves involve a great many numbers. If the machine is to function as a *high-speed* computer, it is clearly essential to limit sharply the necessity for stopping the machine and supplying or removing numbers before the computation is finished; indeed, ideally, the machine should not be interrupted at all during the course of the computation. To provide for a completely automatic sequence of computations, there must be a fully automatic control that can process the necessary instructions (orders) without interruption, and a large-capacity memory that can accommodate simultaneously a great many numbers.

One of the simplest types of memory to visualize is built around a specially designed electron tube, the so-called "electrostatic storage tube", which in construction and operation is similar to a small television tube (iconoscope). At one end there is a network or mosaic of electrostatically charged "spots". Each of these small charged areas can be represented schematically in the manner shown

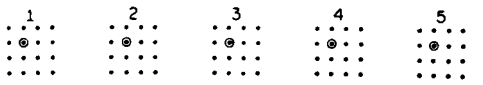

Figure 2. Schematic illustration of the networks of charged points in five identical electrostatic-storage memory tubes. The circles enclose *corresponding* points.

in figure 2. Tube 1 is pictured as a network of 16 points, each point representing a small area of electrostatic charge; tube 2 is identical, and so on. A *number* (series of digits) can be represented as equivalent to a group of charges located at *corresponding* points in the series of networks; for example, the circled point in tube 1 holds the first digit of the number, the corresponding point (circled) in tube 2 holds the second digit, and so on. The arrangement illustrated in figure 2 would in this way accommodate 16 five-digit numbers (16 points in each tube, and 5 tubes). In practice as many as 40 tubes are used, each with as many as 1,024 points (a 32×32 network instead of the 4×4 pictured above); a memory of this size would have a capacity of $32 \times 32 =$ 1,024 forty-digit numbers.

In each tube is a device for producing a narrow beam of electrons that impinges on the network of changed points, and in fact can "scan" this network point by point and row by row. The required deflections of the electron beam are produced by a properly controlled magnetic field, which means that the scanning process can be performed with extreme rapidity. The tubes are synchronized so that at any instant their respective beams impinge on corresponding points of the charged networks; in this way the various digits in a given number can be "sensed" simultaneously and transferred to a register. By proper control of the intensity of the beam, it is possible to arrange for the scanning process to *read*, or to *write*, which means in effect that a number can be transferred *from* the memory *to* the register, or *vice versa*.

The computations required to solve a given problem can be formulated as a list of *orders* that enumerate step by step, every single arithmetic manipulation that must be performed, including all necessary transfers of numbers to and from the memory. The orders are then translated into numbers according to a definite code, and as such are themselves stored in the memory (thereby establishing the link between human control and machine control of the computation). In this sense the memory holds *logical* information as well as strictly *numerical* information; both types of information are expressed as series of digits). Since the orders are now located in the memory, they can be scanned and transferred, so to speak, to the control organ, which arranges for these orders to be performed, one after the other, as they are read.

Virtually every problem of a scope that justifies the use of a high-speed computer will demand a highly versatile control organ. For example, a commonly recurring situation at various stages of the computation requires repetition (iteration) of the same sequence of operations and hence of the same sequence of orders. To illustrate: suppose we have, say, fifty pairs of numbers and that the same set of operations must be performed on each of these number pairs. Obviously, in writing out the orders we cannot afford, as a rule, to repeat the same sequence over and over again the desired number of times because the memory space available for these orders probably would be quickly exhausted. It should be sufficient to list this sequence of orders just once; so the control should be able to "go back" to the beginning of the sequence and repeat as many times as required. In fact, we should generalize this requirement in the following way: normally, the control will take up the orders in sequence as they are entered in the memory; however, at any stage by means of an order that "transfers" the control, it will proceed not to the next order but to any other designated order, and there continue in sequence.

There is another, vital feature of versatility of the control. Where a transfer of the control is required, we should in principle specify beforehand precisely the order to which the control should be transferred; or, if a sequence of orders must be iterated, we should specify beforehand exactly how many iterations must be performed. There are several difficulties here, connected with the fact that very frequently this type of information is not known until the computation has proceeded up to a certain point. If we are to avoid interrupting the computation, such decisions (to transfer

the control) must be made *internally* on the basis of specific orders provided for this purpose. Indeed, even when for example we know in advance exactly how many iterations of a given sequence of orders are needed, there would be no convenient way to terminate these iterations unless a decision can be made internally. All "decisions" of the type required can be reduced to an order providing for *conditional* transfer of control, depending on the sign (plus or minus) of a number held in one of the registers. Suppose, for example, that the first iteration of a sequence of orders produces a number a_1, the next iteration a slightly smaller number a_2, the third a still smaller number a_3, and so on, and we want to iterate until a number, say a_n, is produced that is smaller than a specific preassigned number b. This is accomplished as follows. After each iteration there is an order placing the difference $a_n - b$ in one of the registers, followed by the conditional transfer order. If this difference is positive (meaning a_n is still greater than b) the control will transfer back to the iteration; if it is negative the control will proceed *out* of the iteration to whatever order may follow in sequence. This provision for conditional transfer of control is used repeatedly in most problems. It provides for extremely versatile and fully automatic control of the computation.

Another typical application of the conditional transfer order is this: suppose we know in advance exactly the number of iterations required for a given sequence of orders — say, 30 iterations. Before the first run through the sequence the number 1 is ordered into an empty location in the memory, and just prior to each repetition of the sequence this "counting index" or "iteration index" is advanced by adding 1 to the number already there, so that at the end of each sequence this particular memory location will hold a number exactly equal to the number of times the sequence has been completed up to that stage. Further, at the end of each sequence the difference "29 minus the index" is ordered into a register and then the conditional transfer order follows. When this difference is *positive* (or zero), meaning that 30 iterations (the desired number) have not yet been completed, the control is transferred back to the beginning of the sequence; but after the 30th iteration this difference is *negative*, so the conditional transfer order carries the control out of the iteration and on to the next part of the computation.

We have now discussed the main features of the logical framework of a computer; they are (apart from an input-output device):

(i) An arithmetic organ, consisting of a few registers each to hold a single number (series of digits) at any instant. The arithmetic organ is responsible for performing basic arithmetic operations such as addition, multiplication, division.

(ii) A memory, capable of storing a great many numbers, with provisions for reading and altering its contents as the computation proceeds.

(iii) A versatile control, which responds to a list of orders stored in the memory.

A final comment regarding the *speed* of computation: in an electronic computer all internal operations are purely electronic and therefore extremely rapid. Indeed, the unit of time most convenient in this connection is the *microsecond* — one millionth of a second. A good approximation to the duration of most types of computation can be obtained by allowing about 500 microseconds for each multiplication (or division), and about 65 microseconds for each order (including multiplications and divisions). For example, in a typical meteorological calculation (the 24-hour barotropic forecast) carried out on the Princeton computer, 1,960,000 multiplications and a total of 42,300,000 orders are required. The approximate duration of this computation is therefore $500 \times 1{,}960{,}000 + 65 \times 42{,}300{,}000$ microseconds, or about 62 minutes. This makes it quite clear why an extremely involved and lengthy computation — hopelessly impractical to attempt by hand — may be quite practical for the high-speed computer.

2. A problem

The problem that will now be described has been chosen mainly to illustrate some typical steps leading to formulation of a computation for the high-speed computer. It is in fact a simple problem and requires only a rather small amount of computation, so it is not representative of the type of problems for which a high-speed computer is best

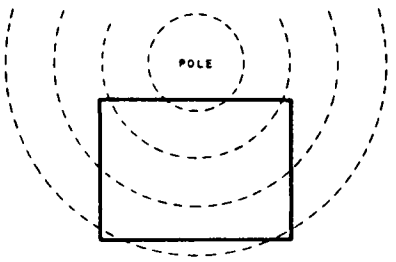

Figure 3. Polar stereographic projection; the dashed lines are latitude circles, the solid lines delineate the region over which the vorticity will be computed.

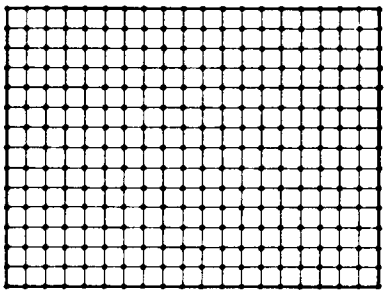

Figure 4. A network of grid points covers the rectangular region of figure 3. The grid shown here has 18 interior columns and 13 interior rows.

suited — namely those that involve extraordinary amounts of calculation.

Suppose we have an analyzed 500-mb map showing height contours. *Problem: to calculate the geostrophic vorticity field that corresponds to the contour analysis.* Let z denote contour height of the 500-mb surface, g the acceleration of gravity, and f the Coriolis parameter, so that the geostrophic wind components are

$$u = -\frac{g}{f}\frac{\partial z}{\partial y}; \qquad v = +\frac{g}{f}\frac{\partial z}{\partial x}.$$

Then the geostrophic vorticity is

$$\zeta \equiv \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} = \frac{g}{f}\left(\frac{\partial^2 z}{\partial x^2} + \frac{\partial^2 z}{\partial y^2}\right), \qquad (1)$$

neglecting here the variation of f in the denominator of the expression for u.

The contour analysis is drawn on a map that represents a plane projection of the spherical earth; this projection will distort true distances. Let Δx be the true distance (on the earth) between two neighboring points and let Δa be the distance between these same points measured on the projection; then $\Delta a = m\,\Delta x$, where m is the *scale factor* for the projection and will usually depend upon the latitude where the measurement is made. To be specific, assume that a polar stereographic projection is used; the scale factor for this projection is

$$m = \frac{2}{1 + \sin\Phi},$$

where Φ is the latitude. Writing $\Delta a = m\,\Delta x$ and $\Delta b = m\,\Delta y$ for distances measured on the projection, the equation for calculating the vorticity from a contour analysis drawn on the projection is, from (1)

$$\zeta = \frac{m^2 g}{f}\left(\frac{\partial^2 z}{\partial a^2} + \frac{\partial^2 z}{\partial b^2}\right). \qquad (2)$$

It will be recalled that the Coriolis parameter f is equal to $2\omega \sin \Phi$, where ω is the angular speed of the earth's rotation.

Figure 3 is a portion of a polar stereographic projection on which a rectangular boundary has been marked out, to delineate the region over which the geostrophic vorticity will be computed. (The contour analysis is not shown.) We now construct a network of grid points (figure 4) covering the rectangular region, such that the distance between grid points is the same in either diretion — say equal to d. In order to specify the location of a grid point in this array, let i designate the *column* in which the point is located (counting from left to right), and j the *row* (counting from bottom to top); thus, $i = 0$ designates the column coinciding with the left-hand boundary, $i = 1$ the first interior column, $i = 2$ the next, and so on; similarly, $j = 0$ designates the row coinciding with the bottom boundary, $j = 1$ the row next above this, and so on.

From the contour analysis, values of the height z must be interpolated (this can be done visually) for each of the grid points. The interpolated height at the grid point whose coordinates are i, j we denote by z_{ij}. Consider a typical group of five neighboring

grid points (figure 5), the central point with coordinates i, j; we wish to calculate the geostrophic vorticity ζ_{ij} at this central point. Returning to formula (2) we see that the vorticity depends upon g (a constant, the same for all points), upon the scale factor m_{ij} at this point, the Coriolis parameter f_{ij} at this point, and the second derivatives of the contour height in both directions at this point.

To evaluate the latter two terms, consider the point half way between the central grid point (i, j) and the next grid point to the right $(i+1, j)$; there, the *first* derivative $\partial z/\partial a$ is approximately

$$(z_{i+1,j} - z_{i,j})/d,$$

where d, as previously stated, is the distance between neighboring grid points. Likewise, half way between i, j and $i-1, j$ the first derivative $\partial z/\partial a$ is approximately

$$(z_{i,j} - z_{i-1,j})/d.$$

The *second* derivative $\partial z^2/\partial a^2$ at the central point (i, j) is then approximately the difference between these two first derivatives, divided by d (the distance between the two "half-way" points); this reduces to

$$(z_{i+1,j} + z_{i-1,j} - 2z_{i,j})/d^2.$$

The second derivative $\partial^2 z/\partial b^2$ is approximated in the same way, by considering the points $i, j+1$ and $i, j-1$ above and below the central point; this gives

$$(z_{i,j+1} + z_{i,j-1} - 2z_{i,j})/d^2.$$

Adding the two second derivatives, as required in formula (2), we get

$$(z_{i+1,j} + z_{i,j+1} + z_{i-1,j} + z_{i,j-1} - 4z_{i,j})/d^2.$$

The terms in parentheses consist simply of the sum of the height values at the four points surrounding the central point, minus four times the height at the central point; for brevity we will denote this group of terms by S_{ij}.

The vorticity at i, j is now to be calculated from the formula

$$\zeta_{ij} = \frac{m_{ij}^2 g}{f_{ij} d^2} S_{ij}, \qquad (3)$$

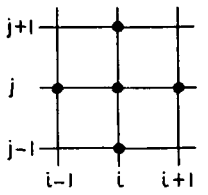

Figure 5. A typical grid point i, j and its four neighbors.

in which g and d are definite constants, m_{ij} and f_{ij} depend upon the latitude of grid point i, j (strictly, upon the sine of the latitude), and S_{ij} depends upon the contour heights at i, j and the surrounding four grid points.

There remains only to find the formulas by which m_{ij} and f_{ij} can be calculated for each i, j. Let r denote the linear distance between the pole and a point at latitude Φ on the polar stereographic projection; the properties of this projection require that

$$\sin \Phi = \frac{1 - (r/2a)^2}{1 + (r/2a)^2},$$

where a is the radius of the earth. If i_0, j_0 denote the grid coordinates of the pole, then the square of the linear distance (on the projection) between pole and grid point i, j is

$$r^2 = d^2 [(i - i_0)^2 + (j - j_0)^2],$$

according to Pythagoras' theorem. The last two formulas provide for calculation of $\sin \Phi$ at any grid point i, j: we need only the values of i and j and the constants i_0, j_0, d, and a.

From the formulas already given, the reader can easily verify that the following computation scheme will yield the geostrophic vorticity ζ_{ij}:

Formulas:

$$S_{ij} = z_{i+1,j} + z_{i,j+1} + z_{i-1,j} + z_{i,j-1} - 4z_{i,j},$$

$$C_{ij} = (d/2a)^2 [(i - i_0)^2 + (j - j_0)^2],$$

$$\zeta_{ij} = \frac{g}{2\omega d^2} \frac{(1 + C_{ij})^3}{1 - C_{ij}} S_{ij},$$

$$(i = 1, 2, 3, \ldots, p \text{ and } j = 1, 2, 3, \ldots, q)$$

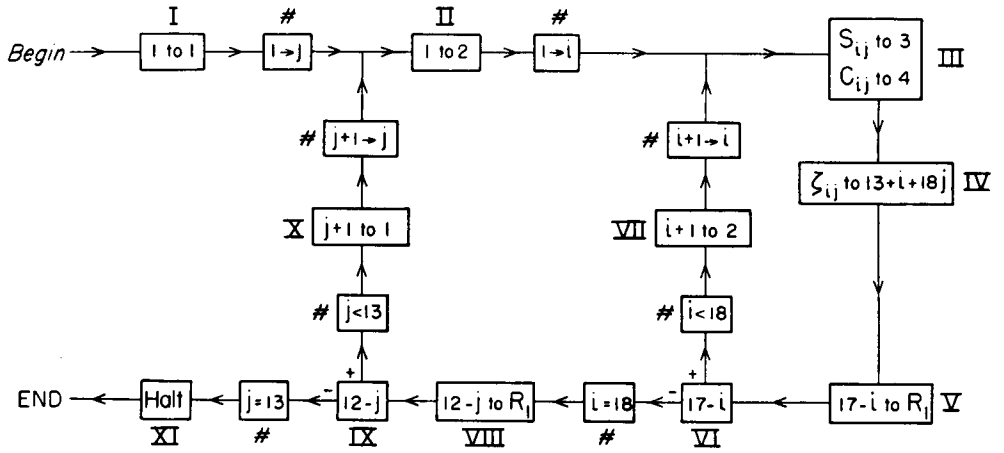

Figure 6. Flow diagram for vorticity computation. The formulas for S_{ij} and C_{ij} in box III and ζ_{ij} in box IV are listed in the program given previously.

Data:

z_{ij} = contour heights at all grid points.
($i = 0, 1, 2, \ldots, p+1$ and $j = 0, 1, 2, \ldots, q+1$)

Constants:

g = acccleration of gravity,
ω = angular speed of earth's rotation,
a = radius of earth,
d = distance between grid points on projection,
i_0, j_0 = grid coordinates of pole,
p = number of interior grid columns,
q = number of interior grid rows.

Roughly 10 multiplications are required to produce ζ_{ij} for one grid point from these formulae. Suppose, further, that something like a total of 100 orders (including nonarithmetic operations, such as necessary transfers of numbers between registers and memory) are required to produce one ζ_{ij}. Allowing 500 microseconds for each multiplication and 65 microseconds for each order, we get an estimate of $500 \times 10 + 65 \times 100 = 11{,}500$ microseconds, or say roughly 0.012 seconds as the time required to compute the vorticity at one grid point. For a grid of 500 points the whole computation would be finished in about $500 \times 0.012 = 6$ seconds — which justifies our earlier remark, that this particular problem is not representative of the type for which the high-speed computer should be used, where the computation time may be of the order of minutes or even hours.

3. A computation

The computation scheme outlined at the end of the preceding section constitutes what is known as the *program*. In order to portray the precise sequence of logical steps by which the control will carry out this program, we construct a *flow diagram*. The flow diagram for this computation is presented in figure 6.

Before interpreting figure 6, it is necessary to agree on a plan for enumerating the memory locations, so that the position of each number held in the memory can be specified. It will be recalled that the various digits belonging to one number are distributed among the memory tubes; each digit is assigned to one tube and the digits of one number appear in corresponding points of the charge networks of the various tubes (figure 1). Suppose each tube accommodates 1,024 points (charges) — so the capacity of the memory is 1,024 numbers — and that these points are arrayed in a square network of 32 rows and 32 columns. Each point in the array is a definite memory location. We enumerate these locations as follows: start in the lower left corner and call this point 0; proceed serially along the bottom row to 31 in the lower right corner; continue with the first (left-hand) point in the second row as 32 and proceed along this row to the end

point 63; likewise, 64 will be the first point in the third row, and so on; the very last point in the array (in the upper-right corner) will be number 1023, so that (since we started with 0) there are 1,024 locations in all.

Turning to figure 6, we note first that the flow diagram consists of a collection of "boxes" connected by branches marked with arrows. Each box (except those marked #) stands for a group of orders necessary to perform the operations indicated by notations within the box. The arrows on the branches connecting the boxes indicate the sequence in which the control will take up these groups of orders. Starting at the place marked "begin", the control proceeds to the first box, numbered I. The notation "1 to 1" in this box is intended to mean that the *number* 1 is ordered into *memory location* 1. The next box (marked #) does not stand for any operations or orders, but is intended to make an assertion as to how the number 1 introduced in the preceding box is henceforth to be interpreted; the notation "1 → j" signifies that we are to regard the number 1 (in memory location 1) as a value — in fact the first value — of the index j. This type of purely logical assertion is useful as an aid in reading and interpreting the flow diagram.

Before proceeding to box II, we remind the reader that i and j identify the column and row location of a point in the grid (not to be confused with the charge network in the memory!) that was marked out on the map projection (figure 4). Interpolated contour heights z_{ij} have been assigned to each of these grid points, and our objective is to compute the vorticity ζ_{ij} at each point. To fix ideas we will refer to a grid with 18 interior columns and 13 interior rows (figure 4), so that i runs from 0 (the left-hand boundary) to 19 (the right-hand boundary) and j runs from 0 (the bottom boundary) to 14 (the top boundary). This grid has $18 \times 13 = 234$ interior points; therefore, the results of the computation will consist of 234 values of ζ. The data required for the computation consist of the values of z at every grid point *including* points on the boundaries; therefore, $20 \times 15 = 300$ values of z comprise the data. (The four corner points are in fact not needed but it is convenient to include them in order that every row may have the same number of points.) In table 1 we have compiled a "storage table", which is a record of the memory locations assigned to the numerous quantities involved in the computation. The 234 values of ζ will be stored in locations 32 to 265, while the 300 values of z are stored in locations 288—587.

Table 1. Storage table.

Memory location	Quantity stored	Remarks (see fig. 6)
0	—	
1	j	Formed in boxes I and X
2	i	» » » II » VII
3	S_{ij}	Computed in box III
4	C_{ij}	» » » III
5—19	—	
20	0	
21	1	
22	$(d/2a)^2$	Used in box III
23	$g/2\omega d^2$	» » » IV
24	i_0	» » » III
25	j_0	» » » III
26	17	» » » V
27	12	» » » VIII
28—31	—	
32—265*	ζ	Computed in box IV
266—287	—	
288—587†	z	Data
588—607	—	
608— ?	Orders	
? —1023	—	

* ζ_{ij} is stored in location $13 + i + 18j$.
† z_{ij} » » » » $288 + i + 20j$.

Proceeding to box II, we have a group of orders that send the number 1 to memory location 2; the following box is an assertion that this number is to be interpreted as a value of index i. In box III we compute two numbers S_{ij} and C_{ij} needed in the formula for ζ_{ij} (note that i and j are the numbers in memory locations 2 and 1, respectively); S_{ij} is sent to position 3 and C_{ij} to position 4 in the memory. We can now calculate ζ_{ij} (box IV), which is sent to position $13 + i + 18j$; for example, $\zeta_{1,1}$ $(i = 1, j = 1)$ is sent to position $13 + 1 + 18 = 32$. The reader can easily verify that this scheme of storing the successively-computed values of ζ in the memory will place the required 234 numbers serially in memory positions 32 to 265 inclusive.[1]

[1] The last three paragraphs of this section give a more detailed account of the computations in boxes III and IV.

At this stage the computation of one of the ζ's is completed — namely, the one corresponding to $i=1$, $j=1$. The flow diagram is arranged in such a way that the ζ's are taken up in the following order: starting with $i=1$, $j=1$ we proceed along the first interior row to the last interior point ($i=18$, $j=1$), then to the first interior point of the next row ($i=1$, $j=2$) and along this row to $i=18$, $j=2$, and so on; the last ζ computed will be for $i=18$, $j=13$. Now the orders involved in boxes III and IV can be used for every ζ, so the control should return to box III each time we pass on to the next grid point. In order to control this iteration we must alter the values of i and/or j (in memory locations 2 and 1) so they will always correspond to the grid point whose ζ is being computed. Further, every time we complete the sequence of orders in boxes III and IV we must determine whether at that stage we are at the end of one of the interior rows ($i=18$), because if we are, one of two steps must be taken: if we are at the end of the *last* row ($j=13$), the iteration must be stopped because the computation is finished; if we are at the end of a row but *not* the last row, we must advance j and calculate for the next row, starting with $i=1$.

The orders that provide for this decision are indicated in boxes V and VI. In box V, the number $17-i$ is sent to one of the registers, designated R_1. Box VI represents the *conditional transfer* order described previously. If $17-i$ is positive (or zero) then $i<18$ and we are not at the end of a row; in this case the control is transferred to box VII (along the "positive" branch of box VI), where the index i is advanced so that we proceed to the next point in the same row. From box VII the control is transferred *un*conditionally back to box III. When $17-i$ first becomes negative we know that $i=18$, the end of a row; we must then determine which of the two steps mentioned above must be taken, that is, we must determine whether we are on the last row. This is accomplished in box VIII, which orders $12-j$ to register R_1, and box IX, which is the conditional transfer order. If $12-j$ is positive (or zero) then $j<13$ and we are not on the last row; in this case the control is transferred to box X (along the "positive" branch of box IX), where the index j is advanced, and then unconditionally to box II, where i is returned to its initial value 1, the first step in starting along the next row. When $12-j$ first becomes negative we know that $j=13$, the last row, and that all ζ's have been computed. (Recall that box IX is reached through the "negative" branch of box VI, so it is entered only when $i=18$, the end of a row.) The "negative" branch ($j=13$) of box IX proceeds to box XI, a halt order.

It is evident that the flow diagram clearly portrays the sequence of operations by which the computation is performed, including the various points at which and to which the control must be transferred. The control cycles repeatedly around the loop (induction loop) formed by boxes III, IV, V, VI, VII, except that every 18th cycle carries it through the "larger" (less frequently traversed) loop formed by boxes III, IV, V, VI, VIII, IX, X, II.

The next step after completing the flow diagram is to write out the orders needed to perform the operations implicit in each box of the flow diagram. The complete list of orders required is the *code*. It will be recalled that the orders, properly translated into digits, will be stored in specific memory locations; referring to table 1, the reader will note that the orders for this computation are assigned to a portion of the memory beginning with location 608. The writer has not prepared the code corresponding to the flow diagram of figure 6, so the exact number of orders needed has not been determined.

There is one further aspect of the computation in boxes III and IV that should be mentioned. In box III the quantity S_{ij} is calculated, and this depends upon the values of z at i, j and at four points surrounding i, j (figure 5). The five values of z needed to compute S_{ij} must be called out of their memory locations by means of appropriate orders that will specify the "addresses" of these z's. Each time we pass through box III we are dealing with a different point (that is, a different pair of values i, j), which means that — as is obvious — a different set of five z's is required each time. Therefore, box III must include a provision for *changing addresses* in the orders that call out the z's; this is certainly feasible because each time we enter box III we know what the current values of i, j are (they are stored in locations 2 and 1), and we also know — according to the previously stated

scheme for storing the z's — that the address of z_{ij} is $288+i+20j$ (see table 1). This is an interesting and important feature of the flexibility of the control: by means of appropriate orders within the code, other orders can be changed whenever such changes are needed to suit the requirements of the computation.

In box IV a similar change of address must be included, in order to arrange for the storage of each newly-computed ζ_{ij} in its previously designated location. According to table 1, ζ_{ij} should be stored in location $13+i+18j$; therefore, before giving the order to store ζ_{ij}, the address in this order must be changed to correspond to the current values of i and j.

Finally, we may note that after S_{ij} and C_{ij} (computed in box III) are used to calculate ζ_{ij} in box IV, they will no longer be needed, so it is not necessary to store these two quantities permanently. In fact, each time S_{ij} and C_{ij} are computed and stored in locations 3 and 4, the values previously computed (and stored there) for the preceding point are automatically "erased".

4. Concluding remarks

The central problem of applied meteorology is weather forecasting in all its phases. One is naturally led to speculate on the extent to which high-speed automatic computers can be brought to bear on this problem. In the first place, it is necessary to repeat a comment made earlier: a computer of the type we are considering excels in dealing with computations that involve large quantities of data and (or), especially, prodigous amounts of arithmetic. The question is then, to what extent are advances in various phases of the problems of weather forecasting at present impeded by our inability to cope with difficulties of the type just mentioned? We will touch briefly only a few aspects of this question.

As a specific example, consider the problem of forecasting radiational temperature changes in the free atmosphere. A great deal is known about the theory of such changes: they are controlled by the distribution of water vapor, carbon dioxide, ozone, and other principal absorbers of long- and short-wave radiation, and by the distribution of liquid water (clouds), as well as the reflectivity and temperature of the underlying ground surface. Numerous graphical, tabular, and other approximate methods have been devised to account for most of these factors, but a more thorough treatment of the problem probably rests on detailed integration of the complete radiational transfer equations. The complexity of these equations is staggering, when one attempts to provide in detail for all of the factors that our present knowledge indicates are important, such as the distribution and intensity of lines in the spectra of the principal absorbers, and their variations with pressure and temperature, to mention only one element of the problem. A direct attack on the problem, attempting to account completely and simultaneously for all known factors, probably would be fruitless. However, it seems likely that a great deal could be learned by using a high-speed computer to integrate the transfer equations under conditions somewhat more complex than can reasonably be treated by means of "hand" computations.

As another example, take the forecast of high-level winds (over relatively short periods, say 24 hours or less) — for example, for use in operation of modern aircraft. Here we are face to face with the problem of predicting the future state of motion of the atmosphere — a problem inherent in virtually all phases of weather forecasting. From the time of Helmholtz, and particularly following the development of the Norwegian school of physical hydrodynamics under V. Bjerknes, meteorologists have on the whole had implicit faith in the belief that the laws of hydrodynamics, diligently applied to the atmosphere, will advance our understanding of the motion of the atmosphere and thereby improve our ability to predict its future state. Certainly it is true that our knowledge of the behavior of the atmosphere is vastly improved as a result of the approach through theoretical hydrodynamics, but because of the overwhelming complexity of the atmosphere, the advances as brought to bear on forecasting problems have been largely qualitative. It must be said that the meteorologist's faith in the efficacy of hydrodynamical methods has not been genuinely rewarded: modern forecasting techniques in most of the world's weather services probably are preponderantly subjective, and for the

most part do not reflect appreciably in quantitative terms, the advances in theoretical "dynamic" meteorology.

In these circumstances we are in fact encountering the difficulties mentioned at the outset of this discussion: strict application of hydrodynamical methods entails an unearthly amount of computation, as well as vast quantities of data, and is far beyond the reach of "hand" computation. Perhaps it is not unreasonable to advocate renewed faith in hydrodynamical methods of forecasting the future state of the atmosphere, because the high-speed computer may provide a means — hitherto unavailable — for helping to develop such methods into an effective quantitative implement.

At present the theoretical foundation of modern "dynamic" meteorology consists largely of classical hydrodynamics, amended let us say, as by Reynolds, in particular to include eddy transport phenomena. We cannot be certain that this framework will suffice ultimately to account for all of the complex phenomena observed in the atmosphere. To some extent it is reasonable to hold the view that modifications or reinterpretations of the classical framework will be suggested in the course of systematic scrutiny of the observations now or eventually at our disposal. However, no matter how thorough or inspired our interpretation of the observations may be, we shall not completely understand the limitations of the classical framework unless we learn a great deal about precisely what can be predicted within this framework; this of course means integration of the (nonlinear) hydrodynamical equations and use of high-speed computers.

In this connection, we should mention the recent and very promising revival of experimental methods of approaching atmospheric circulation problems by means of hydrodynamical models studied in the laboratory (notably the work of Fultz and his coworkers at the University of Chicago). The high-speed computer may prove effective in studying these controlled experiments theoretically, and if so, this coordinated approach may establish a more rigorous basis for testing the theoretical framework of our ideas about the atmosphere.

Acknowledgments. — On several occasions the writer has participated in the work of the Meteorology Group in the Electronic Computer Project at the Institute for Advanced Study, Princeton, and for these opportunities is sincerely indebted to Prof. J. von Neumann and Dr. H. H. Goldstine, and particularly also to Dr. Jule G. Charney. The writer acknowledges gratefully their advice concerning various aspects of this paper.

REFERENCES

There is a very extensive literature on the subject of automatic computers; but since the discussion in this paper pertains specifically to the logical aspects of the electronic digital computer at the Institute for Advanced Study, we cite only the following reports:

BURKS, A. W., GOLDSTINE, H. H., VON NEUMANN, J., 1946: *Preliminary discussion of the logical design of an electronic computing instrument*, Princeton, Institute for Advanced Study.

GOLDSTINE, H. H., VON NEUMANN, J., 1947: *Planning and coding of problems for an electronic computing instrument*, Princeton, Institute for Advanced Study.

For an example of the formulation and programming of a concrete meteorological forecast problem, as well as a report on the results of computation with a high-speed computer, the reader should consult

CHARNEY, J. G., FJØRTOFT, R., VON NEUMANN, J., 1950: Numerical integration of the barotropic vorticity equation. *Tellus*, **2**, 237—254.