Gaines, B. R. (1968). Varieties of computer—their applications and inter-relationships. In Proceedings of IFAC Symposium on Pulse Rate and Pulse Number Signals in Automatic Control, pp. 1-16. IFAC, Budapest.

IFAC International Pulse Symposium
Budapest, April 1967

VARIETIES OF COMPUTER - THEIR APPLICATIONS AND INTER-RELATIONSHIPS

B.R.Gaines

Standard Telecommunication Laboratories,
London Rd., Harlow, Essex -

and Department of Electrical Engineering Science,
University of Essex, Colchester, U.K.

SUMMARY

A major step forward in device technology is being made with the introduction of low-cost integrated circuits and large-scale integration (LSI), the implications of which seem to go beyond those of any previous improvement in computer components. Whereas the transition from vacuum tubes to transistors brought about a tremendous increase in reliability and decrease in physical volume, any increase in physical size was a by-product of these rather than a main effect. What LSI offers is sheer quantity of devices, at a low cost and in a small space - so many devices in fact that the complete processor of a digital computer, or all the active elements of an analog computer, may become a single component.

This paper discusses the impact of LSI on computer systems and computing techniques, and suggests that the major effects will be to make special-purpose computers for specific problem areas very much more attractive, and to blur the distinction between software and hardware - in the sense that computer-assisted design and computer program compiling will merge into a general translation technique from problem specification to both module interconnection (hardware) and variable stored-program (software).

Present theories of computation have been developed either for the minimization of hardware or for the study of programming languages, and are not directly applicable to the hardware/software combination. In particular the inter-relationships between the computer structure and the problem structure are very important considerations to the utility of special-purpose computers, since ease of application is a major requirement in evaluating possible future developments.

It is suggested that the natural system for dealing with the problem of matching computing techniques to problem classes is the theory of maps and their relationships , and in particular the theory of functors over categories of maps developed in homological algebra.

A brief introduction to the application of this theory to computer systems is outlined, which provides a common treatment of the trade-off between sequential implementation of a number of simple operations and parallel implementation of an equivalent single complex operation, and the effect this has on program representation and ease of programming. In particular it provides an explicatum for the ease of operation associated with an analog computer, and suggests an extension of the term 'analog' which preserves this association.

Finally, it is suggested that a natural computer for the real-time problems of navigation and control systems may be based on an incremental digital, DDA-like, arithmetic unit, coupled to a sequential, program-control unit. The pseudo-analog computations of the DDA would be used to generate the complex operations, such as vector rotation and matrix up-dating required in these problems, but its structure need be far simpler than that of previous parallel DDAs because its operations would be varied under program control and used to form an overall computational sequence.

A feasibility study of a modular, programmed incremental computer has been made at STL, and a prototype machine has been developed called the Phase Computer. In the latter part of this paper, the phase computer is described in some detail, and examples are given of a range of basic computations, such as addition, subtraction, multiplication, division, squaring, square-rooting, and rectangular/polar co-ordinate conversion.

VARIETIES OF COMPUTER - THEIR APPLICATIONS AND INTER-RELATIONSHIPS

B.R.Gaines

Standard Telecommunication Laboratories,
London Rd., Harlow, Essex -
and Department of Electrical Engineering Science,
University of Essex, Colchester, U.K.

Introduction

The Impact of Large-Scale Integration

Data-processing and control systems have come to be largely based on two forms of computing technique, epitomized on one hand by the electronic analog computer with its high-gain DC amplifiers and passive networks, and on the other by the general-purpose, stored-program digital computer with its simple processor operating upon binary words and a core-store to hold the lengthy processor control sequence or 'program' involved in any particular computation.

These computers have both been developed to a high level of reliability, small size and low cost, through an evolutionary process dating back at least twenty years. Succesive minor technical advances in materials and components have combined with increasing experience in system design for ease of operation to continually improve their overall performance and utility. One major technical advance significantly improved virtually every performance figure of both types of computer and greatly extended their ranges of application - this was the step from thermionic vacuum tubes to solid state devices.

Now a further major step forward in device technology is being made with the introduction of integrated circuits and large-scale integration (LSI), and the implications of this seem to go beyond that of any previous advance. Whereas the transition from vacuum tubes to transistors brought about a tremendous increase in reliability and decrease in physical volume, any increase in system size was a by-product of these rather than a main effect. What LSI offers is sheer quantity of devices, at a low cost and in a small space - so many devices in fact that the complete processor of a digital computer, or all the active elements of an analog computer, may become a single component.

The benefits of LSI in conventional analog and digital computers are obvious, but not as great as might be expected. The analog computer's reliance on passive components to establish the nature and accuracy of its computations limits the direct impact of LSI since these components are not amenable to micro-miniaturization (except for high-frequency, short-term computation). The digital computer's reliance on an extensive core-store to carry both program and data also limits the direct impact of LSI since the store cost is by far the major proportion of the overall computer cost.

There are regions of computation where extensions of the conventional machine organization will continue to be effective, or become very much more effective. For example, at high frequencies in radio and navigation systems, the improved gain-bandwidth characteristics of modern integrated circuit operational amplifiers makes active filtering techniques based on precision thin-film passive networks very attractive. Similarly in large facility-type computer systems which must be flexible enough to perform virtually any computation, the best utilization of LSI may be to replace core memory with very fast, but volatile, integrated arrays, which can be replenished regularly from disc back-up stores.

Beyond Software and Hardware

These particular examples apart, the real impact of LSI may be summed up colloquially as a swing from 'software-based' computation towards 'hardware-based' computation. The present emphasis on programming a computation rather than fabricating a computer stems from two causes - one being the early development of digital computers when the vacuum tube processor was massive, unreliable and produced vast quantities of heat, so that it made good sense to minimize the processor hardware, and utilize a program stored on drum or in core to achieve computational power - the other being the initial prime need for general-purpose computing facilities shared by large, inhomogeneous populations of users, so that extreme flexibility was the major objective. Now that integrated circuits are available to make the processor small, cheap and reliable, and there is an increasing demand for special-purpose computers dedicated to particular functions in specific equipment, these considerations are no longer universally applicable.

A computation represented by a 'program' or bit-pattern in a core-store may be expressed just as effectively as

the interconnection pattern of a set of circuit modules. Once such a configuration is established the resultant computer is 'special-purpose' in the same way that a particular program for a conventional, stored-program machine is 'special-purpose,' but the step from a problem specification to an interconnection pattern may be as 'general-purpose' as required, i.e. an extensive range of different computations may be established.

It is important to note that special-purpose, hard-wired machines need not lack 'flexibility', in the sense of being able to perform a variety of computations and switch between them as required. It is only necessary that all the computations be included in the problem specification. Since the required computation must be selected in some way, it might be expected that a specification involving extreme flexibility would best be met by a conventional organization involving selection through a program in core-stors !

The wiring of modules to form special-purpose computers enables full advantage to be taken of the hardware offered by LSI, and is particularly attractive in real-time computation where the special-purpose machine will generally be faster, cheaper and smaller than a stored-program, general-purpose machine. Typical problem areas where such machines are of immediate interest are automatic control, aircraft and marine navigation systems, product quality monitoring and air-traffic control. In these areas one may expect the conventional instrument and associated special-purpose computer to merge, probably so completely that any separation between them will be conceptual rather than real. Data-processing techniques will become part of established system technology, and the 'computer' will only rank as a separate unit in the sense than an IF-strip is a unit at present.

Since the design aids for any future system engineering may be expected to contain the equivalent of 'high-level' languages for problem specification, and the appropriate 'compilers' for turning a problem statement into the implementation of a solution, the 'programming' of a special-purpose machine will not necessarily appear different from that of a general-purpose machine today. 'Programming' in itself only implies a clear, unambiguous problem statement, not a sequential implementation, and seems a suitable word to apply to both special-purpose and general-purpose machines. The distinction between 'hardware' and 'software' implementations will decrease in importance, however, since it will be optimization procedures within the compiler which determines the combination of module-interconnection and stored-program to be used in realizing a particular computation.

The Place of Incremental Computing

In these real-time, special-purpose computers, the standard digital computer processor with an arithmetic unit capable of implementing a few simple operations such as data-transfers, COMPLEMENT, ADD, SHIFT, AND, etc., will not necessarily be optimal. To minimize the length of program sequence required and to simplify the problem of implementing a program on the machine, it is best if the elementary processor operations correspond to the natural operations in the problem statement. In numerical data-processing, addition, subtraction, multiplication and division are generally required, but these are insufficient basic operations in navigational systems where trignometric operations such as vector-rotation are of equal importance, or in adaptive controllers and filters where matrix manipulations are also required.

Since these real-time problems are generally generated by dynamical systems specified by differential or difference equations, it would not be surprising if the 'natural' operations were best generated by computational techniques for solving such equations. The computation of trignometric and hyperbolic functions, and the solution of matrix up-dating equations is indeed very simple on the electronic analog computers and on incremental digital computers such as the DDA. As previously discussed, the conventional analog computer has little to gain from LSI, but the 'counting' techniques adopted in the DDA are ideal for realization by arrays of standard digital gates, and multiple DDA integrators will be readily accomodated in a single package.

The 'conventional' parallel DDA in itself, however, is capable only of the complete, single-configuration solution to a particular problem; the sequential decoding of the general-purpose digital computer's 'program' is completely lacking. A DDA set up for a given problem may be regarded as a special-purpose processor without any sequential programming capability, and, on the grounds of coding theory, one would expect it to be wasteful of hardware and inflexible in operation. This has been acceptable in the past because DDAs have been used for the high-speed, continuous solution of differential equations, not for the sampled-data, discrete operations of non-differential, numerical data processing.

The DDA and the conventional digital computer may be seen as opposite extremes in the approach to problem solution - the one adopting an entirely parallel decoding of the problem statement into a single operation, and the other a highly sequential decoding into a multitude of minor operations. The optimum approach would be expected to be somewhere between these two, with an operation set selected according to the distribution of common operations in the problem class to be solved, and sequential statement of particular problems in terms of these operations.

Thus, a very attractive real-time data-processing system might be based on the use of the incremental, pseudo-analog computations of the DDA to generate a

family of computing operations natural to navigational and adaptive control problems, together with the sequencing control operations of the digital computer microprogram to combine the DDA operations in a branching structure which generates the problem solution. In this mode the DDA would be used to solve final-value problems, rather than to give a continuous output, and externally it would appear as a normal arithmetic unit with operations such as ROTATE the vector in registers X and Y through the angle θ, together with more usual operations such as, ADD, SQUARE, TRANSFER, MULTIPLY, DIVIDE, and so on.

A feasibility study of a modular, programmed incremental computer has been made at STL and a prototype machine has been developed called the Phase Computer [1]; this study is reported in the latter part of this paper. Although this machine is complete in itself, however, as are the 'analog computer' and the 'digital computer' and has important applications as a self-contained unit, it is presented here primarily as an example of the possibilities for new forms of computing system with the increasing availability of low-cost integrated circuits and LSI.

Problems of Future Computer Engineering

The combination of incremental digital computing with sequential control used in the phase computer is only one of many possible examples of computing configurations. Further examples of configurations at this level of simplicity are:- the combination of analog and digital computers in a 'hybrid' system [2]; the combination of analog integrators and digital stores and sequencers in Schmid's SADC [3]; combination of parallel binary processing, digital/analog multiplication and analog addition in the Ambilog 200 [4]; the combination of analog filters under analog control in model-reference adaptive controllers [5]; the combination of analog filters under digital control in adaptive line equalization [6]; the use of transfluxor-driven resistive networks for matrix inversion in conjunction with a digital computer [7] - the list could be greatly extended and is growing at an increasing rate.

The number of machines using mixed computing techniques at present, however, is neglible compared with the number of conventional, stored-program, digital computers, processing parallel binary words with standard logic configurations. The extension of the processors of these machines by the addition of multiple hardware accumulators, index-registers, priority-interupt systems, autonomous data transfer units, hardware multiply/divide and variable-length arithmetic units, and high-level language features such as hardware DO-loops, does not essentially differ from the extension by adding analog processing capabilities, for example. The function of an index register is to make it simpler and faster the process data-arrays - the function of an analog section may be to speed and simplify matrix inversion or vector rotation; in all cases the function of the extension is to increase speed, decrease overall cost, or simplify the programming of a particular problem for the computer.

Thus, the structural and functional design of future computing systems will raise problems which, at present, are far more the province of the computer linguist than the electronic engineer. The judicious implementation of closed subroutines which may be called as complete entities in a variety of programs has its parallel in the synthesis of modules with a particular function to be built into a variety of systems. The synthesis of problem-oriented languages and associated compilers has its parallel in the development of families of modules peculiarly suited to navigation or control problems, and the associated design aids for interconnection specification.

In so far as problem areas may be distinguished, there appear to be four of especial importance:-

(i) Computing techniques - the interrelationships between various forms of data-representation and between the means for implementing computations in these various representations.

(ii) Partitioning - the modular elements which can become single LSI circuits and are compatible one with another.

(iii) Sequential/parallel - the relationship between the performance of a computation as a single operation or as a branching sequence of simpler operations, and the hardware/time tradeoffs involved.

(iv) Ease of Implementation- the relationship between the problem specification and the hardware/software of the computer system.

These main areas alone encompass information-processing system function, organization, implementation and utilization, and no distinct separation is possible between them. Although a comprehensive treatment of these problems cannot be given at present, it is possible to establish a theoretical framework for their study, and analyse present computing systems within it. The following section outlines a general theory of computation based on the concept of functors between categories of mappings, and illustrates this with comparative examples from various forms of computer.

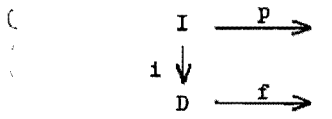
Theory of Computation

Maps Between Operators

Although automata theory provides the necessary terminology and mathematical objects for the analysis and synthesis of sequential circuits to perform defined functions, the main emphasis of its present development is on minimizing the hardware (in some sense) required for the computer to perform a particular computation, rather than matching the computer to a class of computations and maximizing the ease of programming. The natural system for dealing with these problems is the theory of maps and their relationships

[8,9], and in particular the theory of functors over categories of maps developed in homological algebra [10,11,12,13].

The fundamental problem of computation may be stated in abstract - given a map, f, from a domain, D, to a range, R, (the computation), to establish mappings, i, from I (a sub-set of computer input configurations) onto D, and o from Θ (a sub-set of computer output configurations) onto R, such that there exists a map p belonging to P (the set of computer operations or programs) which makes the diagram -



fully commutative. That is, to any object, δ, in D there corresponds one in I and $\delta I^{-1}po = \delta f$, the computer gives the correct result.

Since D could be the set of state/input pairs of a finite automaton and R the set of next-state/output pairs, this definition includes computer simulation of any finite automaton. Equally D ≡ R might be the state-space of a dynamical system and f an infinitesimal time-displacement, in which case p might be the corresponding analog computer up-dating of the state-variables represented by voltage levels.

The requirement for commutativity in the above diagram indicates that a computers function is to simulate another system exactly in its input/output behaviour - that is, in Wiener's terminology, to be cybernetically equivalent to the system. In general, the computer is required to simulate not just one system but any member of a large class of systems, and the transformation, f, may be taken as one member of a category of maps, F. The task of programming the computer is then to find a map from the category of programs, P, onto that of systems, F - T: P → F, together with suitable input and output codings, such that diagrams of the type above are commutative. By regarding the input/output codings as maps in T between the identies for I, Θ, D and R, this can be subsumed into a single map, T, which in the present context is, trivially, a functor from programs to systems.

The structure developed so far is concerned only with the overall transformations implemented by the system and the computer, not with the possible structures of these tranformations - it has been assumed that there is no overlap between the domains of any maps and the ranges of others (except for identities), and hence products of maps are undefined - if p, p' belong to P, then pp' = φ. The crucial feature of computing to be examined, however, is the manner in which complex transformations are built up as (branching) sequences of simple operations, and the structure must be extended to include this.

Consider the computer as a device which can implement any one of a set of operations, S, from and to computer configurations or states. Each operation may be regarded as a set of maps from sub-sets of computer configurations, and the category of all these maps will be denoted by σ. Consider the free semigroup generated by concatenation of maps belonging to σ, Σ - this is the totality of all sequences of operations executable by the computer.

Any member of Σ may obviously be regarded as a program for the computer, but the definition of a program must be wider than this to take in the possibility of branching. A program is defined to be a sub-set of Σ, p, such that if α, α', contained in Σ, belong to p, and -

$$\alpha i_\alpha = \alpha$$

then $\alpha' i_\alpha = \varphi$,

that is, the domains of any maps in p are disjoint - under these conditions p is itself a map. If p consists of only one member of Σ then it is a non-branching program.

Analog and Non-Analog Operations

The map T: P → F from programs to systems may now be regarded as a map from some sub-set of Σ onto F. However consider the non-branching program p ≡ st (s, t belonging to σ), such that -

$$pT = f ,$$

then $f = stT,$

but it possible that -

$$f \neq (sT)(tT),$$

because these two terms are undefined. That is, there may be no operations within the system which are equivalent to those within the computer - examples are obvious, data-fetches, summation of power series to form cosine of an angle, and so on.

If T is a functor not just from some sub-set of Σ but from the generating category of Σ, σ, then it is reasonable to call the computer an 'analog computer' for the systems F. In this case, to every computer operation s belonging to S, there corresponds a system transformation, g, such that - $g = sT$,

that is the computer operation, s, and the system transformation, g, are analogous.

The electronic analog computer is an 'analog computer' in this sense for linear dynamical systems represented in state-form - $\dot{\underline{Y}} = \underline{A}\,\underline{Y}$,

the scalar products of the row vectors of A with the column vector Y each being represented by a summing integrator. Similarly, a digital computer with the (three address) operations ADD, TWOS-COMPLEMENT, MULTIPLY, is an 'analog computer for the operations associated with a certain finite number field, but one with the operations, ADD, TWOS-COMP., SHIFT, is not, although it is capable of

executing the same programs.

The importance of the 'analog' relationship is in its effect upon ease of programming. To program a computer to simulate a system, the exact nature of the system must be communicated to the computer is some coded form. If an arbitrary code is used and there are a large number of possible systems, then the task of the programmer in encoding the problem and the task of the computer system into decoding this into computer operations are both very difficult. If the systems to be simulated have some natural structure so that all system operations may be represented as a sub-set of the free semi-group generated by some basic system operations, then the task of the programmer is simplified by basing the code on the concatenation of symbols for these operations - that is, giving the programmer a problem-oriented language. Similarly, the task of the computer system (its compiler) is simplified if the code is a direct specification of the computer operations - that is, 'machine-code'. If the computer is an 'analog computer' for the systems then these two forms of coding coincide.

One important property of the analog representation is its implication for intermediate results in a computation. The system operation f may be satisfactorily represented by the computer program p. It may happen that f factorizes into two, or more, system operations -

$$f = gh ,$$

and p may probably also factorize, but there may be no factorization of p such that -

$$p = rs ,$$

where $g = rT$

and $h = sT$, unless the computation is 'analog'. Thus, intermediate results in the system may be unobtainable from the computer. This illustrates one of the fundamental problems in replacing an electronic analog computer facility with a digital one. Even though an optimization routine, for example, may be stated in an initial/final value form, the intermediate values may be important in discovering its unforseen defects.

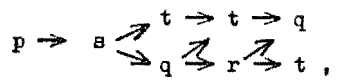
Hardware Implications

There is in general no unique factorization of a set of system operations into a free semigroup generated by basic system operations. The selection of a particular set of basic operations on which to base an 'analog' computer will depend on a number of factors. The cost of the hardware necessary to implement the corresponding computer operations and sequence them appropriately is one. The time taken to perform the operations, whether the overall mean time or maximum time, is another. In small special-purpose machines the sequencing system may be the most expensive item, and minimization of the length of program is a suitable design criterion for factorizing the system operations.

So far the sequencing of computer operations has been treated in a somewhat artificial manner by defining a program as a sub-set of Σ consisting of maps disjoint domains. Thus, if the set of basic computer operations is S:(p,q,r,s,t), then a program might consist of four possible sequences -

psttq, psqtq, psqrq, psqrt

- these maps have different domains and one possible form of sequencing is to test the initial computer configuration to determine in which domain it lies and then implement the corresponding sequence. Alternatively the sequences might be written as a continually branching structure -



in which there is a choice point after virtually every operation.

The sequencing of operations in a branching program can thus be represented as implementation of an operation plus a test to determine the next operation. A test is itself a map from the range of a map representing computer operations to the set of operations itself, and hence constitutes a feedback loop. The basic computer structure may be represented -

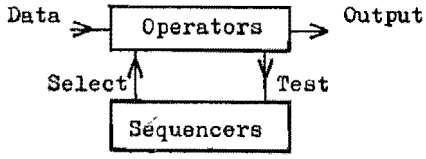


Illustration by Digital Computer/DDA Comparison

Consider the solution of a typical linear differential equation on the stored-program digital computer, for example -

$$\ddot{x} + \dot{x} + x = 0$$

with boundary conditions $x(o) = a$, $\dot{x}(o) = b$. Define a second state-variable

$$y = \dot{x} ,$$

so that the equation becomes -

$$\dot{x} = y , \quad x(o) = a$$
$$\dot{y} = -y-x, \quad y(o) = b .$$

These may be solved reasonably accurately by simple rectangular integration provided time is quantized sufficiently finely, $t = nh$, where h is small, giving the equations -

$$x(n+1) = x(n) + hy(n)$$
$$y(n+1) = y(n) - hy(n) - hx(n).$$

On a large machine with floating-point arithmetic a simple sequence could be written immediately for these equations. On a small machine two sources of difficulty would be apparent - a multiplication

by h is required, and h is very small so that at least double-length precision is required in the additions.

The multiplication by h can be simplified by making h a negative power of 2 - $h = 2^{-N}$ say,

so that the multiplication may be implemented by an arithmetic right shift of N places. This has interesting implications for the multiple length addition, for if the shift is such as to move the number through one full word then the addition operation only affects the lower half of the double-length result, apart from a possible overflow (or underflow - the possibility of negative numbers causes some minor complications which have an effect on the coding of numbers in different DDA registers)into the upper half.

Thus the computation may be reduced to addition or subtraction of the upper registers holding x and y, to or from the lower registers, detecting and storing overflows, and using these to update the upper registers. This is a sequential computation which bears little resemblance to the original dynamical equation, and the computer operations are not analogs of the system operations. When a large number of state-variables are involved the computation is very time-consuming, even though the tricks of multiplication by shift and split double-length working have already speeded it up considerably.

There are two basic techniques for speeding up the computation. The first utilizes the fact that adding in to the lower part of the double-length words, detecting any overflow and setting the overflow increment to the upper word can be made a single operation. This is the basis of the early drum DDAs, to update each integrator by a single operation (although they generally create errors by adding the overflows straight into the upper words in arbitrary sequence so that some results are up-dated when they are added in, others are not). The second technique is to duplicate the hardware required for this operation and update all the integrators together - this is the basis of the parallel DDAs [14,15,16].

The first technique corresponds to taking some sub-set of Σ obeying the axioms of a 'program' and setting up a basic computer operation equivalent to the transformation realized by this progr-am. The second technique is more interest-ing as it demands that parts of the computation be performed in parallel and hence have no sequential interdependence. This requires that the map corresponding to the program may be expressed as a Cartesian product of maps, which is reasonable in the case of analog simulat-ion in the DDA, but may be difficult to determine in other, more general, cases[17].

Conclusions from the Theory

It has been possible in the previous section to give only a superficial outline of a potentially rigorous approach to the design of complex computer systems. It does, however, provide a common treat-ment of the trade-off between sequential implementation of a number of simple operations and parallel implementation of a single complex operation, and the effect that this has on program representation and ease of programming. In particular it provides an explicatum for the ease of operation associated with the analog computer, and suggests an extension of the term 'analog' which preserves this assoc-iation.

The most important conclusion to be drawn from the theoretical treatment is that intuitive considerations of what makes for ease of design and problem-solving in data-processing and control systems do have a firm foundation, and that the system 'costs' in terms of speed, price, flexibility and ease of use may be treated in a common framework. Such a treatment is essential if the hardware/software 'compilers' [18] of tommorrow's computing systems are to become a reality.

This concludes the general and theoretical sections of this paper. In the final section, the Phase Computer is described in some detail. It exemplifies both the modular, LSI-dependent, programm-ing-through-interconnection approach dicussed in the first section, and prov-ides an interesting computing structure for analysis using the techniques outlined in the second section - at the increment-al, or DDA, level the computations may be seen as the repetition of long sequences of simple operations to build up more complex computations, whilst at the 'sequencer', or overall function, level the computations may be seen as providing a variety of complex operations directly available.

The Phase Computer

Introduction

The particular problem which first stimulated the development of the phase computer was the realization of the least-squares smoothing and prediction equations for automatically tracking radar targets [19,20]. Previously these had been imple-mented on a general-purpose digital computer, but it was desired to use them in circumstances where no conventional computer was available, or economically feasible, and the simplest and cheapest, special-purpose equation solver was required.

The tracking equations involve addit-ion, subtraction, multiplication, division, squaring, square-rooting, sine/cosine generation and inverse sine/cosine resolut-ion, so that they encompass a very wide range of arithmetic operations. They may be regarded as Kalman-Bucy equations for optimum system identification [21], and hence any system for their solution has immediate relevance to adaptive control problems. The data-rate in both surveill-ance radars and control systems is fairly low (typically between 0·1 and 100 samples/second) and incremental computing

techniques offered the possibility of solving the equations with very simple hardware. Since the data was already in sampled-form, it was attractive to decrease the cost still further and add to the flexibility of the computer by executing the computations as a sequence of simpler operations.

Apart from its sequential, programmed operation, the phase computer has one other feature not found in previous parallel DDAs, and that is the use of unidirectional counters throughout the computer rather than the bi-directional, up/down counters of the DDA. This feature, which not only decreases the cost but also greatly simplifies the operation, is made possible by use of the 'phase counting' principle in which stored quantities are represented as the difference in the counts contained in two counters. Incrementing both counters does not change the stored value - incrementing one and not the other either increases or decreases the stored value, according to which counter is taken positively (the Store) and which negatively (the Reference). The use of this technique is not uneconomic in hardware, because one counter generally acts as Reference to a number of others.

A block-diagram of the phase computer is shown in Figure One below. It has a conventional four-part structure, consisting of arithmetic unit, program sequencer, auxiliary storage, and input/output channels, but the arithmatic unit is incremental digital, the sequencing section is very extensive and may operate several concurrent programs, and the auxiliary storage in the laboratory prototype machine is realized entirely through a patch-board.

In terms of the abstract structural analysis of the preceeding main section: the counters and modulators are devices for implementing arithmetical operations; the input-control units in the upper section are devices for selecting these operations; the sequencers are devices for implementing a sequence of operations; the input-control units in the lower section are devices for selecting this sequence; and the differentiators are devices for detecting conditions on which to branch the program sequence.

A detailed description of these elements and their functions is given in the following sub-sections, followed by a description of their use in simple processors for various computations, together with some examples of input/output devices for analog and numeric data.

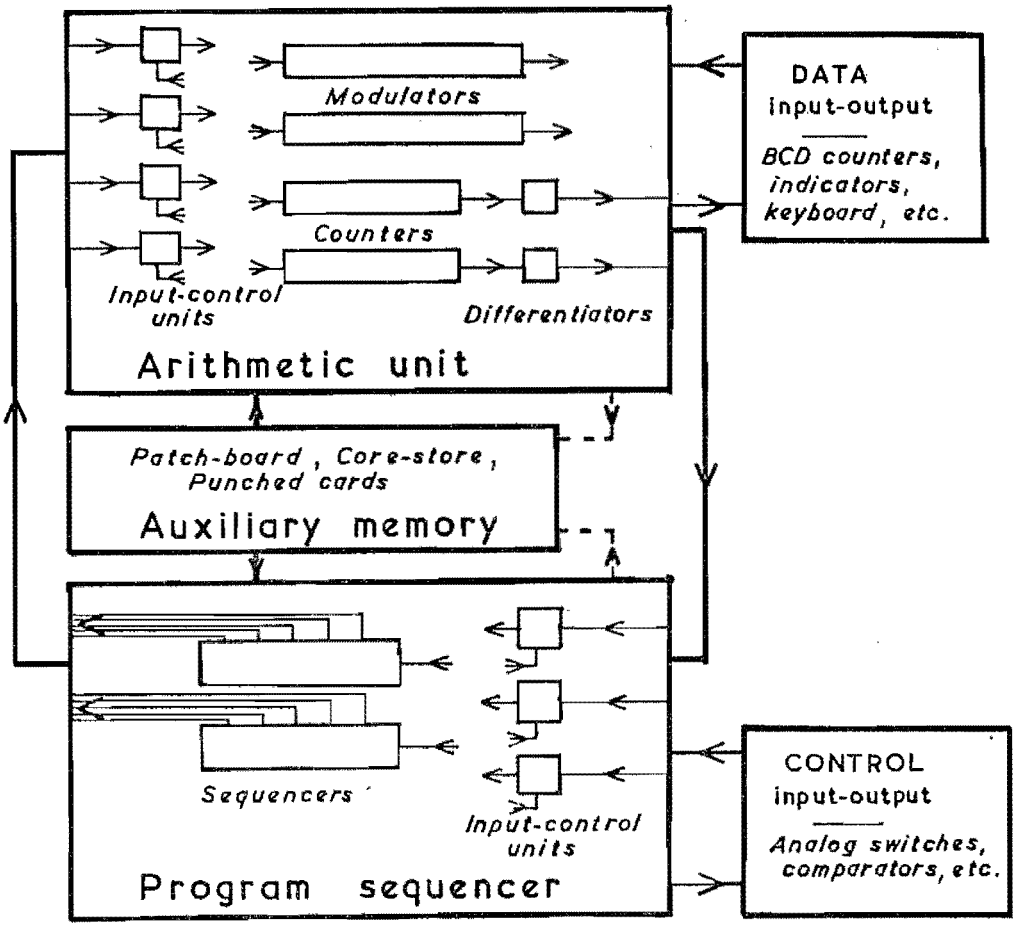


Figure One - Phase Computer Schematic

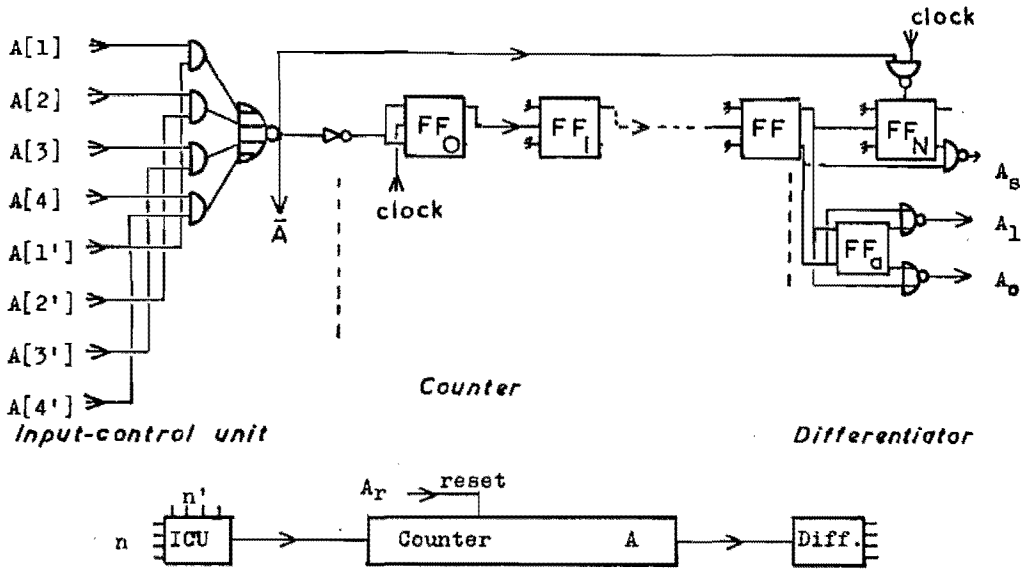


Figure Two - Counter, Control and State-detection

Counters and Modulators

The active storage registers of the phase computer are unidirectional, synchronous, binary counters which, at a clock pulse, increment by unity if their INPUT line is ON, and reset to zero if their RESET line is ON. Although the overall counters operate synchronously, they may be internally asynchronous, as shown in Figure Three (reset omitted from logic diagram).

One peculiarity of the counters is that they have $2^N + 1$ states, achieved using N+1 flip-flops, so that the fractional binary number stored in the counter may take the full range of values from zero though unity, but the zero state and maximum state are identical in the counting cycle which has only 2^N stages. This enables counters used as storage registers to take a complete range of values, but does not lead to an additional state causing difficulty in some computations.

If the proportion of ON logic levels on the INPUT line of the counter is considered as the input variable, then the counter may be regarded as a discrete version of an analog integrator, in that its stored count will be proportional to the input times the period of integration. The counter alone, however, lacks an output in the form of its input - the integral is available as a binary number rather than an incremental sequence.

An incremental sequence in which the proportion of ON logic levels is equal to the fractional binary number stored in a counter is obtained, as shown in Figure Three, from a modulator element which adds (through full-adders) the number in the counter to that in a register (D-type flip-flops) and outputs the overflows.

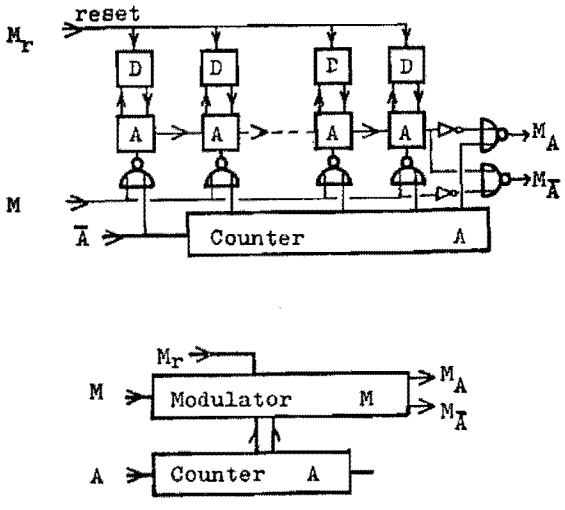


Figure Three - Modulator

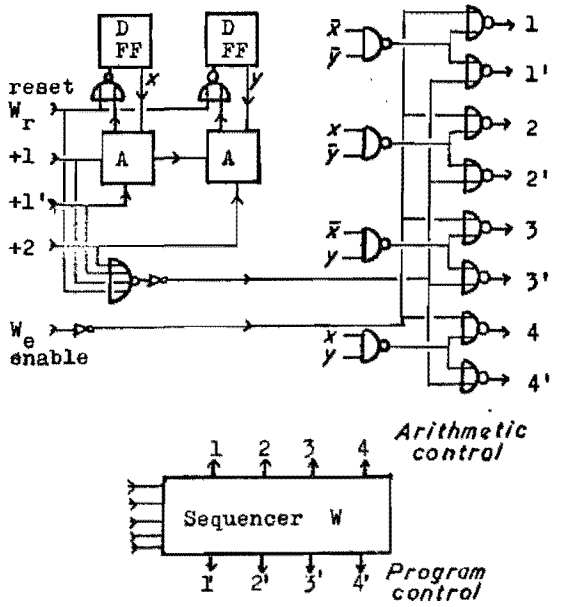


Figure Four - 4-way Sequencer

In order to maintain accuracy when the count in the counter is changing, the quantity added into the register is the mean of the present count and the coming next count (closed trapezoidal integration). The modulator has input lines for controlling additions to the register and for resetting it.

Sequencing Computations

By connecting the outputs of modulators to the inputs of counters, pseudo-analog computing loops may be set up in the arithmetic unit of the phase computer and used to generate operations which are difficult to realize with parallel digital arithmetic. In general, however, the loops used will be far simpler than those of conventional DDAs, the majority of counters generating ramps rather than nonlinear functions, and the basic computational operations realized in this way, such as multiplication, division, addition and subtraction, are combined in programmed sequences to yield more complex overall behaviour.

The essence of programming in all computers is the control of the interconnections between processor elements by logic levels on CONTROL lines. The input-control unit, shown in Figure Two, is a gate enabling processor connections to be controlled - it implements the function:

$$A = A[1]A[1']+A[2]A[2']+A[3]A[3']+A[4]A[4'],$$

and may be regarded as a digital selector switch in which the INPUT line A[j] is switched through to the output if the CONTROL line A[j'] is ON alone. In some applications two, or more, CONTROL lines may be required for each input.

Input-control units placed at the inputs of phase computer counters enable them to be interconnected with modulators in several alternative patterns, any one of which may be selected by activating the appropriate CONTROL lines. A sequencer unit, shown in Figure Four, is used to activate the CONTROL lines corresponding to different interconnection patterns in a programmed sequence. It is made up of a small register, to each of whose states there corresponds an OUPUT line which is ON when the register is in that state. These OUTPUT lines are connected to the counter CONTROL lines so that different states of the register give rise to different computing configurations.

An adder and gating enables the state of the sequencer to be reset to zero or incremented by any amount (generally unity for normal operation and two for a branch) - reset taking precedence over incrementing. All the OUTPUTS of the sequencer are turned OFF for the single clock pulse at which it is incremented or reset. A second set of OUPUTS is also made available for control of input-control units to sequencers and these are not turned OFF as the sequencer changes. None of the sequencer outputs is ON unless its ENABLE input is ON.

The signals causing the sequencers to change their states, corresponding to steps and branches in the program, are obtained from digital 'differentiators' which give an ouput for the clock pulse following that in which the counter enters a certain state - in particular, the zero (A_0), and mid-range state (A_1). The outputs of the differentiators are used to change the states of sequencers when certain counters accumulate pre-determined counts during a computation.

The use of these various elements and their interconnection to perform computations is best made clear by example, and the following section describes phase computer configurations for a wide variety of basic arithmetic operations.

Unsigned Arithmetic Operations

A small phase computer processor might consist of three counters, A, B, C, one or two modulators, L, M, a 3-way sequencer, W, and associated input-control units and differentiators. The sequencer would be used to switch the inputs to counters either ON or OFF, or to the outputs of modulators, and the connections associated with each state of the sequencer can be specified by the signals at the counter inputs. The outputs from the differentiator will be used to change the state of the sequencer or reset counters, and this program control can be specified by the changes that may occur, and their causes, in each state of the sequencer.

Thus any particular computation may be specified completely by a table of the following form:-

Sequencer -	State -	W_0	W_1	W_2
	Reset -	0	A_0	A_0
	Advance-	P	B_0	0
Counters -	A input-	0	1	1
	B input-	0	1	1
	C input-	0	0	1
	C reset-	P	0	0

This table corresponds to data-transfer, or reproduction of the quantity stored in counter B in counter C, assuming that counter A initially is at zero and that the computation is initiated by a pulse, P. In the zeroth state of the sequencer, W_0, all counter inputs are OFF - this is the passive state. The pulse P resets counter C to zero and advances the sequencer to its first state W_1 in which the inputs to counters A are both ON (written as '1'). When counter B overcounts to zero, the differentiator output B_0 causes the sequencer to advance to state W_2 in which the input to counter C is also ON. Finally the sequencer returns to the passive state W_0 when counter A overcounts back to its initial state of zero.

The overall effect on the counters in this computation can be determined from the quantities transferred in during each state of the sequencer. If the fractional binary number initially in counter B is β, then the quantity transferred in to bring

the count in B to zero is $1-\beta$. Thus when the sequencer advances to W_2, A contains $0 + (1-\beta) = 1-\beta$, B contains $\beta + 1-\beta = 1 \equiv 0$ (complete cycle), and C still contains zero. During the sequencer state W_2 the quantity required to return counter A, containg $1-\beta$, to zero is β. Hence finally A contains $(1-\beta) + \beta \equiv 0$, B contains $0 + \beta = \beta$, and C contains $0 + \beta = \beta$.

This computation illustrates the phase counter principle, for if the quantity stored in B is taken to be the difference in counts between counter A and counter B then this is invariant provided the inputs to the two counters always receive the same signal - it will be noted from the table that this is so. Since counter A starts and finishes in zero, it has no initial or final effect on the stored quantity which may be taken as the fractional binary number in B.

The table may be abbreviated -

	W_0	W_1	W_2
W_r	0	A_o	A_o
W(+1)	P	B_o	0
A	0	1	1
B	o	1	1
C	0	0	1
C_r	P	0	0

Data Transfer - B → B, C

If the initial resetting of C is omitted then this becomes addition of the quantity in B to that in C -

	W_0	W_1	W_2
W_r	0	A_o	A_o
W(+1)	P	B_o	0
A	0	1	1
B	0	1	1
C	0	0	1

Addition - B → B, B+C → C

Subtraction is a simple variant on this in which C counts during W_1 rather than W_2 -

	W_0	W_1	W_2
W_r	0	A_o	A_o
W(+1)	P	B_o	0
A	0	1	1
B	0	1	1
C	0	1	0

Subtraction - B → B, C-B → C

If C initially contains γ, since the count added whilst the sequencer is in W_1 is $1-\beta$, the final total in counter C is $\gamma + (1-\beta) \equiv \gamma-\beta$.

For multiplication, B is connected to the modulator, M, and its input is OFF throughout the computation so that its state does not change. The ouput of M, M_B, is a sequence of logic levels in which the proportion of ON levels is equal to β, the fractional binary number in B. This is connected to the input of C between C entering zero and A returning to zero, a period proportional to the quantity in C, γ - so that the final quantity in C is $\gamma\beta$:-

	W_0	W_1	W_2
W_r	0	A_o	A_o
W(+1)	P	C_o	0
A	0	1	1
B	0	0	0
C	0	1	M_B
M	0	0	1
M_r	P	0	0

Multiplication - B → B, C x B → C

Division is realized in a similar fashion, except that the connections to A and C are reversed, so that during the sequencer state, W_2, the quantity γ has to be supplied to A from the source M_B, and hence the quantity transferred to C during this period is γ/β:-

	W_0	W_1	W_2
W_r	0	A_o	A_o
W(+1)	P	C_o	0
A	0	1	M_B
B	0	0	0
C	0	1	1
M	0	0	1
M_r	P	0	0

Division - B → B, C/B → C

Squaring the quantity in B offers the first example of a modulator being used to generate a nonlinear function through connection to a changing counter. The output from the modulator M during W_1, whilst B counts from its initial value up to zero would be -

$$\int_{\beta}^{1} u\,du = (1-\beta^2)/2 ,$$

if the modulator and counter inputs were ON together. By connecting the counter input to a source whose proportion of ON logic levels is 1/2 (a toggling flip-flop), the number of clock intervals for B to return to zero is doubled, and so is the above integral. This quantity is counted into A, so that at the end of W_1 A contains $1-\beta^2$, and the remainder to be counted in during W_2 is β^2, which also enters B:-

	W_0	W_1	W_2
W_r	0	A_o	A_o
W(+1)	P	C_o	0
A	0	M_B	1
B	0	1/2	1
M	0	1	0
M_r	1	0	0

Squaring - B^2 → B

Taking the square-root of the quantity in B is performed in a similar fashion

except that inputs to counters and modulator are interchanged between sequencer states W_1 and W_2. The net effect is that the quantity transferred to A to bring it back to zero during W_2 is both equal to β, the initial quantity in B, and to the square of the final quantity in B, λ^2 say - hence $\lambda = \sqrt{\beta}$:-

	W_0	W_1	W_2
W_r	0	A_o	A_o
W(+1)	P	B_o	0
A	0	1	M_B
B	0	1	1/2
M	0	0	1
M_r	P	0	0

Square-root - $B^{1/2} \longrightarrow B$

Rectangular/polar co-ordinate conversion requires an additional modulator, L, coupled to counter A. The vector (x,y) in Cartesian co-ordinates, whose components are stored in counters B and C respectively, is to be converted to (r,θ), polar co-ordinates stored in the same locations. First, during W_1, 1-y is transferred to A, and then counters A and B are cross-coupled with the modulator output M_B at the input of A, and the inverted modulator output $L_{\overline{A}}$ at the input of B. This may be represented by the differential equations -

$$dx/dt = y$$
$$d(1-y)/dt = x$$

so that $\ddot{x} + x = 0$,

corresponding to rotation of the vector (x,y). This ceases when A returns to 0, in which event the y-component has become zero and the x-component in B is equal to $r = (x^2 + y^2)^{1/2}$. The quantity accumulated in counter C during W_2 is proportional to the angle of rotation, θ, in units such that one right-angle corresponds to unity in the counter, and the conversion from radians is accomplished by counting from a source in which the proportion of ON logic levels is $2/\pi$ (obtained from a modulator with constant, hard-wired input):-

	W_0	W_1	W_2
W_r	0	A_o	A_o
W(+1)	P	C_o	0
A	0	1	M_B
B	0	0	$L_{\overline{A}}$
C	0	1	$2/\pi$
L	0	0	1
M	0	0	1

Rectangular/Polar Co-ordinate Conversion

$(B^2 + C^2)^{1/2} \longrightarrow B$, $\tan^{-1}(B/C) \longrightarrow C$

Signed Arithmetic Operations

The preceeding sub-section has dealt with arithmetic operations on positive quantities only. Signed numbers are represented in the counters in twos-complement form with the sign bit reversed so that zero quantity is represented by the mid-range value in the counter. The sign bit is not connected to the adder in the modulators, but is used to gate the normal and inverted outputs of the modulators to give an additional output proportional to the magnitude of the twos-complement binary fraction in the counter; for counter B connected to modulator M this output is designated M_B^*.

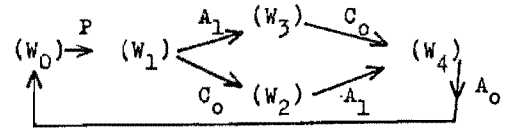
Computations with signed quantities are generally a little more complicated than with unsigned since branching is generally necessary to take the signs into account. These may be accomplished, in the basic computations considered so far, by means of a 5-way sequencer using both ADVANCE +1 and ADVANCE +2 inputs.

For example, multiplication is done by a configuration corresponding to the table:-

	W_0	W_1	W_2	W_3	W_4
W_r	0	0	0	A_o	A_o
W(+1)	P	C_o	0	C_o	0
W(+2)	0	A_1	A_1	0	0
A	0	1	1	1	1
B	0	0	0	0	0
C	0	1	1	1	1
D	0	$\overline{B}_s$	M_B	$M_{\overline{B}}$	B_s
D_r	P	0	0	0	0

Signed Multiplication - B x C $\longrightarrow$ D

If the quantity in C is positive then C_o comes on before A_1 and the sequence is $W_0\ W_1\ W_2\ W_4\ W_0$, whereas if it is negative A_1 comes on first and the sequence is $W_0\ W_1\ W_3\ W_4$, so that the branching structure is -



The computation may be checked out by considering each of the four possible sign combinations in turn - for example, it the quantity in B, β, is negative, and so is that in C, γ, then the sign bit of B is $B_S = 0$, and the path of computation is $W_0\ W_1\ W_3\ W_4$. The quantities accumulated in counter D are - 1/2 in W_1, $\beta\gamma$ in W_2 and zero in W_4, so that δ, the final quantity in D is given by -

$$\delta + 1/2 = 1/2 + \beta\gamma + 0,$$

and thus $\delta = \beta\gamma$. Similarly the other possibilities may be checked.

Addition, subtraction and data transfer are performed with the 3-way sequencer configurations outlined previously, except that B_1 replaces B_o because of the modified twos-complement representation. Division, squaring, vector rotation, and so on, are all readily performed with signed quantities, but the configuration

used will vary with the overall computation, since several operations on different registers will generally be performed at the same time.

Input/Output Devices

Since most applications of the phase computer are in real-time data-processing, control and display systems, input and output has been an important design consideration and facilities for BCD/binary conversion and analog/digital conversion fit naturally into the structure of the machine.

Any counter in the computer is potentially capable of being set directly with a binary number, and also of being read out directly from the condition of its flip-flops. It is more convenient, however, to use special interface counters for binary input and output, and transfer from these to the required register using a counting sub-routine. The internal counters, used in computations, can then have a very simple structure with a minimum of connections. Interface counters having a BCD representation may then be used for automatic decimal to binary, and binary to decimal, conversion.

Input and output of analog data on many parallel channels is performed in the phase computer by a single digital/analog convertor, plus analog track/hold and comparator units for each channel. The D/A convertor generates a ramp by outputting the quantity in a cycling counter in analog form, and this is compared with sample analog inputs to control the counting of other counters which act as input data registers, or sampled by track/hold units at the output which are themselves controlled from counters acting as output data registers [1].

Summary and Conclusions

The topics treated in this paper have ranged through the impact of LSI on computing systems, the development of an integrative theory of computation that enables hardware and software problems of all forms of computer to be treated in a single, coherent framework, and a specific example of a novel form of computer in which incremental arithmetic and programmed sequential computation are combined. It is convenient in this final section to take these topics in the reverse order.

The phase computer is a modular, digital, data-processing and control system, in which many of the advantages of analog and digital computers are combined through the use of an incremental digital arithmetic unit under program control. The incremental processor enables pseudo-analog computing loops to be established so that complex operations, such as division, square-rooting, rectangular/polar/hyperbolic co-ordinate transformations, and so on, may be performed with the same speed and simplicity as simpler operations, such as addition and data transfer. The programming facility enables complex computations to be performed as a sequence of elementary operations, and hence overcomes the inordinate hardware demands of previous parallel incremental computers.

The speed of the incremental processor in performing simple operations is less than that of a parallel digital arithmetic unit because of the counting technique used, and the disparity becomes greater with increasing precision in computation. This factor is very much decreased in computations where a number of operations may be carried out at the same time. At present, with a 4 Mcs clock rate, operations can be carried out with 10-bit precision and accuracy in times of about 250 microseconds, and with 12-bit precision and accuracy in about 1 millisecond. These speeds and accuracies have proved ample for a large class of data-processing and control applications, and enable advantage to be taken of the economy and simplicity of phase-computing.

Applications investigated to date include: area coverage systems for aircraft navigation based on VOR/DME beacons; direct readout of latitude and longitude for hyperbolic marine navigation based on differential transmission paths; sweep generators for PPIs in surveillance radar systems; marker/vector generators with rectangular/polar co-ordinate conversion facilities in PPI consoles; automatic tracking of radar targets by least-squares smoothing and prediction for small radar systems; identification of process parameters for purposes of adaptive control; and on-line product quality monitoring with reject-rate prediction.

Apart from its intrinsic advantages, the phase computer is also of interest as an example of a computing system in which the range of basic operations includes those peculiarly required in specific problem areas, such as vector rotation in navigation systems, and matrix up-dating in identification systems. The relationship between the computer operations and the structure of the system it is programmed to simulate may be formalized algebraically in terms of natural mappings between categories of maps. Such a theory of computation gives a formal explicatum for the concept of an 'analog computer', and makes clear the reasons why a computer bearing this relationship to a class of systems is simple to use.

The phase computer is also of interest because it is an all-digital, modular system based on modern integrated circuits. There are only five basic modules in the computer - counters, modulators, input-control units, sequencers and differentiators. They may be fabricated with standard integrated circuits and utilize fully the high packing density devices, such as multi-bit counters and adders. By their simplicity and uniformity of structure, and their few interconnections, the modules and module-systems are eminently suitable for large-scale integration.

The phase computer is, however, only one particular example of the possibilities opened up by modern, low-cost integrated

circuits and the impact of LSI on computer organization. In an accompanying paper [18] R.A.Shemer discusses more general arithmetic and sequencing units, and their application to both programmed DDAs and to conventional general-purpose digital computation. Such modules may operate in several modes and implement various computing techniques within the same computational cycle. Indeed, the distinctions between techniques become arbitrary in such a structure, since parallel DDA operations may always be treated purely algebraically as series expansions of the functions generated and are most conveniently analysed in this way.

It should be possible at the present state of knowledge to dispense with hard and fast distinctions between analog, digital, incremental, and hybrid, computers, and so on, and treat the technology of data-processing and control in a unified and coherent manner. The main obstacles to progress in this direction are our lack of detailed knowledge about computing techniques in themselves - of their relationships to hardware, and particularly to what constitutes a viable module both from the viewpoint of the computer engineer and from that of the component engineer - and last, but by no means least, of the interrelationship between the hardware/software combination which is the computer, and the structure of the systems it is required to simulate and the problems it is required to solve.

Acknowledgements

I am grateful to many colleagues in ITT for discussions and suggested applications of the phase computer - in particular to Mr.P.L.Joyce of STL who built the first laboratory prototype machine.

References

[1] Gaines, B.R., Joyce, P.L., Phase Computers, Proc. 5th Int.Congr.AICA, Lausanne, August 1967.

[2] Korn, G.A., Korn, T.M., Electronic Analog and Hybrid Computers, McGraw Hill, 1964.

[3] Schmid, H., Sequential Analog-Digital Computer (SADC), AFIPS FJCC 27(1) 1965.

[4] Hagan, T.G., Treiber, R., Hybrid Analog/digital Techniques for Signal Processing Applications, AFIPS SJCC 28 1966.

[5] Donalson, D.D., Kishi, F.H., Review of Adaptive Control System Theories and Techniques, in Modern Control Systems Theory McGraw Hill, 1965.

[6] Lucky, R.W., Techniques for Adaptive Equalization of Digital Communication Systems, Bell Syst.Tech.J. 45(2) 1966.

[7] Karplus, W.J., Howard, J.A., A Transfluxor Analog Memory Using Frequency Modulation, AFIPS FJCC 26(1) 1964.

[8] Clifford, A.H., Preston, G.B., The Algebraic Theory of Semigroups, Vols. I and II, AMS 1961 and 1967.

[9] Ljapin, E.S., Semigroups, AMS 1963.

[10] Eilenberg, S., MacLane, S., General Theory of Natural Equivalences, Trans. AMS 58 1945 231-294.

[11] Cartan, H., Eilenberg, S., Homological Algebra, Princeton U.P. 1956.

[12] Northcott, D.G., An Introduction to Homological Algebra, Cambridge U.P.1962

[13] Freyd, P., Abelian Categories, Harper Int.Ed.1966.

[14] Bradley, R.E., Genna, J.F., Design of a One-Megacycle Iteration Rate DDA, AFIPS SJCC 21 1962.

[15] Mitchell, J.M., Ruhman, S., The TRICE - A High Speed Incremental Computer, IRE Nat.Conv.Rec. Pt.4 1958.

[16] Goldman, M.W., Design of a High Speed DDA, AFIPS FJCC 1965 27(1).

[17] Conway, M.E., A Multiprocessor System Design, FJCC 1963.

[18] Shemer, R.A., A Hybrid-Mode Modular Computing System, IFAC Symposium on Pulse Techniques, Budapest 1968.

[19] Simpson, H.R., A Method of Processing Radar Data to Obtain Position, Velocity and Turn Information, RRE Memo.1924 1962.

[20] Mellberg, K., Experience from Automatic Tracking in ATC, IEE Conf.Pub. No.28, Symposium on ATC Control Systems Engineering and Design.

[21] Kalman, R.E., Bucy, R.S., New Results in Linear Filtering and Prediction Theory, Trans.ASME March 1961.

B. HRUZ: You mentioned the problem of identification of a system in automatic control. In what do you see the advantages of the DDA or incremental techniques in identification modelling systems if it is borne in mind that the use of A–D and D–A converters is needed and this complicates the system. This is interesting in respect to speed, cost, reliability, etc.

B. R. GAINES: This is the kind of point which I think we would very much like to bring up in discussion, because your question raises a large number of quite distinct points on the interface between a computer, Digital to Analogue and Analogue to Digital, with incremental techniques. Of course we have applied them to quite simple controllers, PID controllers, where the output has been some kind of stepping device, a valve driven by a stepping motor, and we have had a digital actuator. And again, the cheapest form of A to D converters are counting techniques which ideally interface with the system. So I do not think one should complain that the interface to the computer presents difficulties, in fact one of the main reasons for using this kind of computational technique is that we can get very much cheaper interfaces.

Within the computer itself, identification is one of those words once again which cover a multitude of things. There is identification e.g. in terms of steepest descent techniques. And here, one has within the computation generally a large number of multiplications and has a feedback loop outside, of a performance criterion with which one is tending to match it. We have had comparisons between various techniques, the analogue ones, the incremental ones and so on, and once again the incremental techniques show up very well as soon as one gets away from the stage where the GP machine is capable of counting within a computation.

Within the GP machine it always comes to a factor of time. If you have got the time to go through a program loop to do the computation necessary for steepest descent, then this is more often than not the best way to go about it. It is quite an extensive computational loop, the more inputs you have got, the more parameters you have got, the more times you have got to go around the loop, and once you get up to something like ten parameters then you really are taking up of a lot of GP time. This is the kind of level at which one wants to go to parallel techniques. If you go to parallel techniques you have got a choice between trying to put several general purpose processors on to it. Each of these will require a program and are rather expensive. If you go to analogue elements, the steepest descent identification requires analog multipliers which are expensive and unreliable.

In the other areas of identification, there has been no attempt so far to use identification methods in a discrete state space, say with models of processes as Markov transitions, probabilistic transitions between states. This has all been written out in theory, but has never been implemented in practice, except, on some simulations which in themselves have never got very far because it is a very expensive thing to simulate. This is the kind of thing which one can now do, using inceremental techniques, and provided the elements are cheap enough. We have it performing for example in our laboratory, where we have put special hardware onto a GP computer, so the arithmetic unit of the computer can look like a special purpose DDA element. This is for simulation purposes, so we can, in fact, simulate networks of several thousands of integrator elements, without building more than one special purpose unit. This really is the problem area of what advantages does one gain. Are there other ways of gaining the same advantages which are going to come first, or are we in fact avoiding the question of these ways at the moment with DDA-s, and just going to find that there is a bigger way coming along with variations on GP machines? Or are we really on to something which is going to become increasingly important within control systems? I think this should form part of this afternoon's discussion.

After a recess, the Chairman, J. Hatvany, asked B. R. Gaines to introduce the Plenary Discussion.

B. R. GAINES: The first point which I would like to raise is probably relevant to the members of the Technical Committees of IFAC who are here. When we run a conference, then by the very nature of the editing process we have a very lop-sided view of the whole field. If I had put in a paper for this conference on GP computers and their applications to automatic control, Dr. Hatvany would have returned it to me and said "It is a very pleasant paper, but completely irrelevant to the conference" and equally if I had put in a paper in which I said that we had been working with DDA-s for about ten years and we tried to apply them to control systems and it had been completely fruitless, he would have said "Hard luck, but this is a very negative paper" and returned it to me. So the only papers we tend to get, are by people who are proselytes for DDA-s and other such systems. And I wonder whether I should take the opportunity of giving the papers which I never sent in because I knew they would be rejected? And trying to put the case for the GP computer, for the poor old analogue computer which we all tend to sneer at, and for the various techniques which are appearing nowadays.

We all introduce the DDA through Large Scale Integration, and say: "Oh, the DDA-s are going to make a come-back because LSI's going to make integrated circuits very much cheaper." This of course, is affecting almost anybody building any system whatsoever, and it is reasonable to look at the GP computer and say: "Well, it has got a very simple processor and a massive core store, and all we are really making at the moment is volatile memories, so they are going to have to go on using those

magnetic cores, and they do not know how to make multiprocessor machines and so they are not going to be able to take very much advantage of Large Scale Integration." In a way this is true, but equally there is a very large number of people with vested interests in the GP computer. And they are really the ones who generated Large Scale Integration. They can not be doing this just to pass themselves out of the market.

I was in New York at the IEEE International Convention recently, and from Texas Instrument there was a device coming out which was a read-only memory, with a cycle time of 20 nanoseconds, in which at present they have got about a thousand bits on a slice but they are aiming by 1973 to be supplying in production quantities, one with about 300 thousand bits per slice. Now this is a pretty substantial program size, so that for many of the applications we have been discussing, applications to automatic control, where programs are fixed, we are going to be able to use this for quite a number of jobs. Let us take for example a PDP 8 computer. The processor there, using Texas 74N series, takes up about 200 packages. This is probably equivalent to about 1000 to 1500 gates. So in three of four years' time that is going to be 3 or 4 packages coupled to this type of read-only memory, and we have got a complete computer, a very powerful machine, which can have as many data registers, as many volatile registers as we would like, in a few slices, at a very low cost. This can be a very powerful device. One important advantage of this read-only memory, is that the programming is done externally by blowing internal links, by in fact burning through the metallization. So they are offering a GP component imposing no real constraints on the computer manufacturer, putting out something which has got a very large market, and costs very little. We certainly will see very cheap GP machines becoming available, and we have got a whole army of programmers, and quite a lot of experience in using these machines. So if we are putting forward DDA-s this is one thing we are fighting against.

In another area, at the Spring Joint Computer Conference there were several papers on hybrid techniques, and the most interesting ones were using D/A converters as computer elements in very much the same way as we have been proposing incremental modules. They have been taking the D/A converter, which once again is very amenable to Large Scale Integration, and seeing how to interconnect large numbers of these to form a computing system. This is a device where you put in a digital number on one channel, and get what is really a variable analogue weight, in fact a digitally controlled potentiometer, where you can put analogue signals through several of these, and control the gates digitally. Adage, for example, are using this in a very nice graphical display which plots a complete line in 4 microseconds and can plot about 5 thousand lines flicker-free on a display and rotate them in 3 dimensions, or more if one really wants it. So here, once again we have an example of a computer technique.

What is tending to happen with these is exactly the same as happened with DDA-s. People are calling them by all kinds of odd names so as to pretend to their own companies that they have invented something new. They are generalizing them and building up a complete system, and they are in fact restricting the applications by trying to use these elements throughout the system. If one, in fact, examines the variety of computer techniques available now, then it has become very very much larger during the last few years. The GP computer, the analogue computer have been coming down in price, and equally there have been a large number of new techniques.

I think "new" is probably a bad word here, for every one of the techniques which one sees and which one gets excited about, one can count back to about 1950, and can find there. In fact, if one looks at the history of the analogue computer and the digital computer and the DDA, one arrives back inevitably at the Second World War, one finds the Bush mechanical differential analyzer being used for ordnance calculations, one finds ENIAC transformed into EDSAC under the influence of von Neumann, so that the stored program digital computers have been generated in about 1946, and one also finds the operational amplifier which is the basis of the analogue computer, being made available as a high stability reliable component at about much the same time. If one looks at the growth of these up to about 1960, the growth patterns are very similar. From then onwards we have got an ever increasing growth, according to at least an exponential law, particularly in the stored program digital computer, especially as it went from scientific to commercial use. The analogue computer seems to have fallen off fairly level, and the DDA looked as if it was just going to droop away into nothingness.

Looking through the previous conference proceedings, one can find that the conferences specifically on incremental techniques are around the 1960-s, so that there was about an eight years' gap, eight years in which there has been very little communication among people working in this field. To some extent, one might say that this has indicated a lack of interest in DDA-s. But the lack of communication is really because they have been applied largely to military uses. However, when examining the possibility of incremental techniques having some place in future systems, we must be aware of the competition here, and certainly a strong competition from a new generation, it is not the 5th generation or the 4th generation, it is an offshoot virtually — a micro-program special purpose computer but using the standard GP techniques, taking advantage of Large Scale Integration. And this is certainly a very strong competitor in the control field and in putting forward the advantages of the DDA one should also really consider exactly the same problems proposed to this type of machine and look at the concurrent advantages.

I think there are two major points, which come out of this. At present we have a number of

isolated people working on DDA-s for specific problems, writing up some aspects and this is the body of literature. Now within a particular company with a particular problem to solve, this literature is not generally accessible. We have very little education at our universities on incremental techniques, we also should have a lot more on GP ones, but from the point of view of getting a job done quickly, the DDA just is not available. If a person who happens to know how to use it, is available at the same time as the problem comes up, where it is the best technique, then this is the ideal way of getting a good search for using incremental techniques. But one example of how the DDA suffered because of this kind of work is, I think, with Litton Industries. They pioneered the commercial applications and they had a major contract for a DDA in an aircraft navigational system. But in getting this contract they had to get more staff in, and the person they put in charge of this project had in fact just been working on GP machines. He looked at the problem and said: "I do not know a thing about DDA-s, but I know how to solve this with a GP machine", and in fact for that particular contract they used a GP machine throughout, and since it was about the biggest contract so far on DDA, this is the kind of factor which has killed the machine.

I think if we are going to use DDA-s we must certainly have the equivalent of high level languages. We must have a problem-oriented language, which instead of giving us bit-patterns of course, at the end of the program, gives us a specification for interconnected modules. Equally, this is extraordinarily difficult to see coming about. Here, once again, the comparison is with a GP machine, and one knows that programming for GP machines is generally very inefficient, so the fact that if we do this for DDA systems we would be equally inefficient, is not a cause for undue alarm, perhaps. Yet it is very difficult to see at the moment any technique for presenting a wide range of problems and getting out a hardware system, a hardware compiler, in fact. Equally, if we are going to have a wider use of the incremental system, then we have to introduce it into a more extensive body of literature, and we have to present examples of problems solved which are of general interest. I think another criticism which has been made of the conference as a whole, is that although we are working under the auspices of IFAC, on automatic control, we have tended to neglect specific control problems and concentrated on the particular problem of incremental computer techniques. It would be interesting to know how much success people are having in applying DDA-s to specific control problems.

J. L. SHEARER: I would like to put some questions which, may or may not be provocative, but might, I hope, lead to some interesting discussion. There seems to be a great reluctance on the part of most people who have used digital computers to get mixed up with what I would call multiple arithmetic units. Now this is essentially what you have been talking of Mr. Gaines, and in that category of arithmetic units you have been including DDA and incremental computer techniques. So the questions I would like to pose to the experts—and incidentally I consider myself to be a neophyte in these questions—are these:

What is the role of reliability or if you like, lack of reliability in the development of multi-arithmetic unit computers?

What is the situation with languages and software, or perhaps I should say, what are the limitations of language and the limitations of software with regard to multi-arithmetic computers?

D. W. RIGHTON: In my firm we are designing aircraft control systems for use in, shall we say, the late seventies. This has to be a digital type system, because digital equipment is going to be very cheap in those days. We are having to decide whether this is going to be an incremental-type system or a GP system. And the reason why we have tentatively picked the GP system is that we do not know what type of control law we shall be synthesizing, and there the GP gives us a flexibility that the DDA type of arrangement does not seem to give. I wonder if there is any comment on this?

B. R. GAINES: Let me take the worst case. If one says one has got a control problem, and one is going to use a GP machine for it, and one does not know the control laws, then there is obviously no guarantee whatsoever, that given a specific control law which is put up as the one you want to use, when one puts it into a GP machine the computation is not, in fact, going to take so long that one just can not use the machine. So I think the first point here is that even though the GP machine to a very large extent, can do almost any computation one requires of it, in a real time application, such as a control system is, one has got to worry about timing. So the flexibility in fact, does not guarantee that the machine is capable of doing the specific control task.

D. W. RIGHTON: We have of course some experiences which allows us to estimate the probable complexity of such a control. We hope that we shall have improved our techniques of control by the time this equipment is put into service. There are two things. We are not quite certain of the machine we are going to control and so there are going to be requirements with regard to parameters. Then we hope that one's knowledge of control theory will have advanced sufficiently in the hardware design time, to make the type of control law which we are using, in fact, the specific control law which we are now using, wrong.

B. R. GAINES: In radar tracking we started off using a steepest descent technique which was a least mean square linear identification technique, for we have got various plots on a radar screen and we are trying to predict where they are going as a track. This is a computation one can write down with arithmetic and work out how long it takes. It is being applied on a GP machine as a subroutine, and the GP machine is capable of handling something like 200—300 tracks. It is being applied on an incremental machine as a piece of hardware, and

if you want to track several planes you have several hardware boxes.

This is a case where, in fact, our knowledge of radar tracking techniques is considerably increased. One thing about the error function for instance: in a radar system one has far better information along the radius factor than one does about azimuth, about angle, so if your error is not a circular component it is quite a distinct ellipse, and this ellipse is rotating in the XY coordinates as the axes go round. Equally, an aeroplane gives one trouble in tracking when it does a turn, because it rapidly moves away from the track, (If it just slows down or speeds up in the same track, this is O.K.) So there is an error function for detecting when the aeroplane does a turn: in terms of cross track errors, how far away is it, at right angles to the track. One in fact generates three entirely different coordinate systems, the radar ones, which are rotating, the XY ones, which are the ones to display, and the cross track ones. Now if, to do the mathematics, you work out an optimum equation for the tracking, taking into account all these effects, then on a GP machine you could track 2 or 3 planes instead of 300 using this. On the incremental one you would have to multiply hardware by about a factor of five to do it. I think this is very tipical of this. If you change the parameters in your laws, that's fair enough, but the GP machine and the special purpose one can quite easily cope with this. If you entirely change the laws and get a very much better technique, then it consists of increased hardware in one case, or decreased time in another, and I think this point of flexibility is not so much a defect of the DDA in itself but in our current thinking about the machine. Equally, the digital computer in real-time applications is vastly less flexible as we see it. Typically again, in the air traffic control we have a very big ATC system in Britain, which as it was built was extensively modified, so it has never been completely built and it has given a lot of trouble. The same applied to air traffic, air ticket reservation systems, so I do not think things are black and white, I think it is definitely a management attitude that if you buy a GP machine for one job if you buy it for control and it's not much good, you can always turn it over to accounting.

But you can not do that with DDA, and this certainly is not a competitive point. The question remains, however, of whether it is a desirable one?

J. HATVANY: My first point is on this question of reliability. I am afraid I shall be repeating myself to those who were there this morning at Section I. I would like to make this point, which I think is a very very important one in defence of incremental techniques (if indeed they need defending). And that is, that they tend to permit one to build a computing system with many homogeneous functional units which can be switched for diagnostic purposes to do the same thing: count. Because they are mostly counters. In the schemes which were shown by Dr. Gaines we could see that, and certain integrators again, are a set of homogeneous functional units. Now I think from the point of view of reliability, this permits a self-checking of these types of machines, which cannot be achieved by any other known machine type because everywhere else all functional units are different, or at least there is a very great variety of them. There is not the same possibility of letting the whole machine work as one set of parallel counters or 2 or 3 sets of parallel counters and checking them bit by bit for coincident progress as though they were just arranged in a simple square matrix.

On the question of software, I think that the software future (and I am certainly not alone in thinking this), will tend towards adaptivity as far as automatic control is concerned, towards the automatic machine evolution of control algorithms from the machine study of the results of datalogging. That means that you will get an output from this machine study of the situation and from the point of view of programming what that output should be, there does not really seem very much difference between getting it to output a sequential program for a central classical processor machine, or getting it to output blueprints of if you like the printed wiring, of a structure and putting that structure in. In fact I even have a hunch that if once such a program is written, the latter will prove a simpler program. But that of course is for the future, since even the first, although it is said to be a very much better known set of situations, is very far from being written.

H. ORLOWSKI: I would like to express my opinion of the comparison of GP and incremental computers. As far as the speed of computation is considered, I believe the incremental computers in industrial control systems have no special advantages, because the actual requirements are not serious for present day GP computers. The really important field of application I have seen in the small discrete schemes, used in single control loops, where there is a necessity of correction, multiplication or generating of certain function. The single loop in such scheme gives no possibility for the use of GP computer on the time-sharing principle. In these cases, GP computers would be obviously less convenient than the incremental ones, because of both cost and reliability.

Please note, that I have talked of industrial applications, and not displays, rockets and others for which the comparison could differ.

T. H. THOMAS: I would like to add another point to the thought of the last speaker and that is the old, well-known problem of engineering design time. I am more interested perhaps in process control than in the much more difficult problems that were discussed here earlier. In process control there are a number of lightly interrelated or unrelated loops and the problem is to have a device which can be applied simply, by relatively unqualified people, to the control of these loops. An optimum solution would certainly be a device which was extremely complex and would need hill-climbing routine to find the correct parameter itself. But until we reach this stage, the wired device, the DDA, which is extremely simple, which has one or two knobs on the outside and which can be applied for a wide range of processes to give,

shall we say, near optimum control, is a far better solution than something which even were it as cheap, requires advanced engineering skill in finding the correct algorithms.

I. ALEKSANDER: I would like to make a rather vague comment because I feel that perhaps the subject under discussion is not central to my knowledge. But it seems to me that as control problems become more and more difficult, one tends to rely more and more on storage in computing systems, whatever they may be. In the applications talked about earlier, the reason that the GP computer seems attractive is because one can store the data of curves of some kind for which there is no clear mathematical correlation. So I think that for development in the future, the choice is neither incremental nor general purpose but a close look at the sort of information which one has got, information on a control situation. The extraction of the relevant, or perhaps statistically relevant part of this information require a lot of storage to develop a control function which can then be implemented in more storage-type equipment. But perhaps that is another problem, it is the next step.

J. L. SHEARER: Why is it that all general purpose computers have only one arithmetic unit?

J. HATVANY: I should like to attempt an answer, if I may. I think, the answer is no. There are a number of general purpose computers with several arithmetic units. Well, one of the first I know of was the Bull Gamma, and then there were others, and now some very big computers are being built as multiprocessor computers. There is the Atlas which has two processors and now of course the Solomon computer. The Iliac I think has 32 in the first stage and can have 64 processors, so I think the multiprocessor computer has been invented. But I do not know whether that was what you were thinking of, because these are general purpose computers.

J. L. SHEARER: Actually that was what I was thinking, but it is one thing for the invention to be made. However, it seems —at least to a neophyte like me —that these machines have not been successful in everyday use.

G. J. MOSHOS: There has been quite a number of computers which have been designed with a number of arithmetic elements. One of the main problems of this class of computers, is in the analysis of how to rewrite programs. One must analyse, so one can decide which arithmetic unit to use. This needs a great deal of analysis in our language translators, as a result of this they have not been too popular. I think the trend however, and this also touches on the questions of reliability that you have put, the trend has been towards more processors. And the problem is one of assigning a task rather to the processors, in which each processor has an arithmetic unit and has also separate logic capabilities. The trend in largescale computers is one of multi-processors for the purpose of stand-by capability. That is, we must be able to detect errors when they occur, we must be able to keep the computer's operation up, some parts of the computer up, 24 hours a day, 7 days a week. We must be able to reconfigure the computer, and when a piece of it goes down, then the supervisor, the executive programme of the system, must be able to reconfigure the computer, so that we can utilize that part of the computer which is up, for two purposes. To carry the work load, the part of the work load which must be done, and also to aid in the localizing of the errors of the part of the computer that is down.

B. R. GAINES: I think the question about multiprocessor machines is a very relevant one here, because it brings us right back to the structure of data processing systems, whether they are for control or any other purpose. It is quite clear, one can look at the historical reasons for the development of a single processor, and one can see this just in the fact that we started with vacuum tubes and these are rather unreliable. So we had the minimum hardware in the processor, and concentrated on making a really good drum system to back it up with a large store. But this no longer applies. Quite certainly, I think it would be fair to say that the general purpose multiprocessor machine has been pretty unsuccessful. Though there are examples of very large, general purpose facility type machines which have been succesful. But if we take the CDC 6600, the multiprocessors there are largely for input-output capability, in fact, for things which have got to be done in real time, asynchronously, which involve far slower data rates than the central arithmetic unit wants to deal with on an autonomous basis by several processors. But in a way these are special purpose processors, and are configured for a particular input-output job, as it comes up. The Iliac 4 is most certainly not a general purpose machine, in this sense. If one looks at the justification for Iliac 4, the only real region where it does justify itself is the field of weather forecasting, where one has a set of pretty complex partial differential equations which have got to be solved in a three-dimensional space. One knows what the connectivity of the space is, one can configure a system, which is about 1000 times faster than an equivalent general purpose machine doing the same job. But once again it is a special purpose application. Certainly it looks very much as if in the case of the general purpose machine, where one says virtually: "I do not know what this machine is going to do, it is just the fastest general facility I can offer, I am not going to restrict people in any way" that one is forced right back to the single processor machines. The reason being, that if you do not know your computational mix, you do not know your type of problem, then in fact, restructuring the computer purely on a software basis, which is of course, done by the programmer and not by the computer manufacturer, is the optimum way out. And if we actually had this hardware compiler, we have been talking about vaguely, a thing where you do not put a specific problem in, (obviously if you are designing hardware, you put a range of problems in e.g. you say, I want to solve a navigational problem, or a certain type of control system problem), if we really put in to the input

of that compiler the general purpose facility problem, then probably what we should get out of it is the standard general purpose computer we have today. However, it does look very much as if we have got to move into some more specialist direction. What we are looking for now is not just one complete problem, because everything we want to do has separate problem classes, for control certainly, like a specific type of data processing hardware, equally for navigational systems, or for communication systems. Message switching is another example, where multiprocessors are being used to achieve reliability very successfully because one knows exactly what one wants in a message switching system. And one can look at the likely failures and design around them. But it is, once again, a special purpose.

H. SEIEDRAZY: Mr. Kalyayev was speaking today of a multi-incremental computer. What is an efficient application for this type of computer, and what is the advantage, over the usual DDA?

А. В. КАЛЯЕВ: Разработка параллельных вычислительных структур является одним из наиболее перспективных направлений развития вычислительной техники.

Особенно многообещающими могут быть параллельные вычислительные системы, состоящие одновременно из трёх типов однородных структур: однородных универсальных вычислительных сред, однородных цифровых интегрирующих структур и из однородных аналоговых вычислительных сред. Однородные цифровые интегрирующие структуры могут быть построены на основе универсальных цифровых интеграторов, окруженных элементарными коммутирующими элементами. Успехи микроэлектроники позволяют уже в ближащее время создать цифровой интегратор совместно с экстраполятором и двумя сумматорами в виде единой миниатюрной твёрдой схемы. Ещё более простыми и миниатюрными могут быть выполнены элементарные коммутирующие ячейки. В результате оказывается возможным сконструировать весьма компактную и в то же время очень гибкую однородную интегрирующую структуру. Коммутацией такой структуры может управлять универсальная однородная логическая вычислительная среда, которая наряду с функциями управления будет также решать логические и другие задачи, не реализующиеся в однородной интегрирующей структуре.

С помощью подобных параллельных вычислительных систем, состоящих из однородных интегрирующих структур и однородных универсальных вычислительных сред может решаться очень широкий круг задач, включая задачи управления, навигации и цифрового моделирования в реальном масштабе времени.

Подобные однородные структуры обладают разнообразными благоприятными свойствами. В частности, они позволяют обеспечить высокую точность и скорость работы за счет использования быстродействующих точных интеграторов. Однотипность цифровых интеграторов и коммутаторов, входящих в однородную структуру, позволяет в необходимых случаях резко повышать надежность за счет дублирования и троирования решающих блоков, а также обеспечивает взаимозаменяемость элементов структуры при выходе некоторых узлов из строя. Важным свойством однородной структуры является возможность перестраивать при необходимости программу в процессе работы. Следует также подчеркнуть перспективность использования рассматриваемых структур в самонастраивающихся и самоорганизующихся системах. К числу достоинств их расширения за счёт простого наращивания однотипных решающих блоков, технологичность узлов, которые могут производиться в массовом порядке на основе схем микрорадиоэлектроники, и, наконец, малые габариты и вес.

Y. LUNDH: The title of this afternoon's discussion is "The place of incremental techniques in the computing systems of the future". I think it is going to be very difficult to predict what will be the future. And I think it would be very difficult to reach a nice formula — to give an answer such as DDA, or DDC a general purpose computer, with such and such data. I think we all know that there can be no such ideal answer, because there never is such an ideal answer to design problems. But what I think is useful, is to know all the various clever methods which are commonly known by these names, DDA etc. They could be useful when you have to design a specific dataprocessing system to do such and such. I myself have done work on digital frequency techniques and we came up with a much simpler solution to that specific problem, than if I was restricted to the general purpose computer. Of course, if as Dr. Gaines said, you do not know your problem, then the general purpose is the most likely thing to solve it because if you have to install a machine and you do not know your problem; but a problem which is solved that way, will be very unlikely to be a very optimum solution, I think. So if you know a specific problem I think it is a great strength to know as many methods as possible — and methods that come under the general heading of this Symposium could be quite useful, not to speak of the methods which come under the heading of a general purpose computer.

There is one other point that I would like to make and that is that general purpose computers can, of course, be used for almost anything, but it will be a sequential job, and you are limited by the memory capacity, etc. For many solutions you will find that the engineering that you have to spend in the software of these systems, the programming problem, could have been much simpler, if you used the techniques available today, in integrated circuits. Integrated circuits reduce the engineering of implementing a specific logical, sequential problem. And that fact, I think will be a factor which places increased importance on the use of hardware solutions, as against the software solutions, because it is easy to implement a specific logic circuit when you have these nice integrated circuits.

One more point, if you want to buy a general purpose computer today, to solve a specific problem, what you pay for is—I would like to be corrected if I am wrong—but I think most of what you pay for is the memory. And that really is the limiting factor, I think, for the economy, the speed for any specific purpose, so that you can implement more, specialized functions in the instruction repertoire. Or else you can use special electronic arithmetic devices, let us say for second order interpolation. These could simply be used as a hardware extension. So these factors, I think, together will point in the direction of hardware, versus software.

J. HATVANY: Mr. Orlowsky has pointed out that general pur pose computers are now fast enough to do anything required in a process. I do not agree with that. If you take a very simple case, for example an oil refinery, and consider a number of flow-meters. Each flow-meter provides some 4—500 pulses/sec. The pulse-number has to be corrected for temperature, viscosity, linearity. In order to correct for temperature you have to correct the local temperature, shift it, multiply it, you have an a + bx first term correction, at least, and all that you have to do for say 100 pulse meters, of flow meters. That alone is enough to completely clog down the order of computers which otherwise would be perfectly sufficient to run that size of plant. If you simply have very small, cheap, special purpose units doing these continuous computations, multiplication, shifts, addings, etc. on-line, you can use your computer for, what somebody has said, it is for, decision making. Now I would like to take issue with Dr. Gaines, if I understood him right. And that is, I certainly see a trend, and I do not agree with him completely. Over Iliac he is probably right, but I see a trend towards multiple arithmetic. Perhaps not quite in the sense in which I have said may be I did not express myself right, but the normal arithmetic units of the bigger computers are more and more tending to be multiple. In that they have several multiplication units, several division units, run-time shared as part of the central arithmetic. That is a trend towards having specialized hardware, and there is absolutely no reason to limit it to those instructions which are usually in our computers. Why should there not be a special integrating unit in the arithmetic part, why should there not be several, working on several items, and there you see the two fields more or less coalescing.

One thing that has been said here several times over, is that there is a promising future trend probably towards substituting prewired special structure for software. Surely that is to some extent also a part of the trend, at least compatible with the trend which is called firmware, of having what we now use as software, that is to say algorithmic languages, autocodes, symbolic languages, pre-wired in large read-only stores. But why use this only for the interpretative languages, why not use it for the problem orientated languages, why not use it for the solution of the problem itself? And if you do that, you have got a special purpose computer, or at least in your computer you have got a special purpose part. So I think in this way, the trends do, somehow, show a certain parallelity and even convergence, provided that we do not make the categories too hard and fast.

J. L. SHEARER: I gather that the answer to the question may be something like this:

1. From the point of view of reliability it is probably wise to have more than one arithmetic unit in a computer.
2. It appears the language, the software, is the real reason why multiple arithmetic unit computers have not really come to the fore as yet.

G. G. MENSHIKOV: I wish to give an example of a problem which is more convenient for the general purpose computer than the incremental. The discussion today is based on steady opinions. Thus, the discussion might have been programmed before. A computer could realize questions, answers, and do it better than we. But this is only possible for a general purpose computer.

G. KAPS: One field of application for computers is time shared computer control, known as DDC. What would you say? Are there any advantages of incremental special purpose units of the type you described in your paper, compared with normal general purpose computers with respect to switching and computing speed and reliability?

B. R. GAINES: This is a problem of current interest in fact, which is being looked into by the Ministry of Technology at Warren Spring in Britain. Take a simple control algorithm that is being applied at present to say, distillation columns, batch reactors and so on, a simple PID control algorithm. It seems possible on a computer about the size of the PDP 8, to control about 40 batch reactors with the system. If you look at your system cost, a major part of it is in the interface, certainly in interconnections. Another feature concerning these reactors is that the run-up and run-down procedures, which are not control laws but open loop procedures, decision making ones quite often, which have to be followed, are equally, important in the computation and one can not just replace the controller. One has got to replace the at present wired logic, which is around the analogue controller, for these particular procedures. On the application of incremental techniques we have done a study typically to realize the Proportional-Integral law, which is the most common one, around the fairly noisy plants in distillation columns. We need 3 counters, about 9 bits in length, and a sequencer unit of 16 steps, plus an associated logic. The cost of this comes out to about 30 or 40£, the components for a control loop. At the Ministry of Technology they have looked into the possibility of multiplexing this type of controller around a number of control loops, and they have got a central processor working in a non-incremental mode, couples to several incremental units at the interface, and they are hoping to control something like 100 loops with this special purpose computer.

The position is at present that in the comparison they are all coming up neck and neck, and there does not seem to be much difference. If you want to do less than about 20 loops, then certainly it is rather cheaper to have a thing which gives you single loops. If you want to do only 3 loops, which is what a small firm may require, then

having a general purpose machine which is always totally committed to this operation, because it is a real time operation, there is no chance of foreground-background use of a general purpose machine, is evidently uneconomical. Certainly the special-purpose hardware is the best thing to go for. We feel that if you have got to about 40 loops, or up to 100 loops or more, then if you are implementing these simple algorithms then the general purpose machine, provided it is used in a correct way, which means it has quite a lot of hardware interface, is the optimum solution. Tipically the hardware interface is determined by facts like that one can not drive the stepping motors on the valves at more than about 300 pulses a second. Now from the output of the computer you can drop out binary numbers quite easily. You have got to drop the binary numbers into counters, and to count down into the actuators. And this is quite a substantial amount of hardware.

In some of the systems you see in looking at direct digital control, the amount of hardware at the interface as it stands, is competent to perform the control law completely, because it is a counter, there are logic gates around it, and you could put a controller straight into it.

If you now looked at the possibility of incremental adaptive control, then the special purpose machines begin to look a lot more favourable. As soon as one goes to say a model reference, adaptive control system, the general purpose machine now has a lot more computing to do, you cut down the number of loops and if one looks at cost-effectiveness, putting more hardware around, a special purpose one looks more attractive. But the main problem with this is that if one looks at the state of the art in direct digital control, apart from the big petrochemical companies, there is very little commercial application. If one actually goes into it, goes to a firm who, say, are making resins, and have got a large formalin plant, you find that the control system is virtually unobservable. There is hardly a transducer on it, and it is virtually uncontrollable, there is nowhere where you can enter to change the process very much. If one is really going to try to optimize these systems, we need a lot more work on transducers, we need a far greater understanding of the control problem in its own right, before we start even talking about optimum control systems. So we are very much constrained at the moment to doing what chemical engineering would like us to do; to provide them with cheap hardware for realizing the control loops, they have got at present.

The real advantages of the hardware seem to come in the next stage, when applying more complex control, and this is really where the control problems come in. I think one important point here is that control theory itself is not generated in a vacuum, if one looks to algorithms which use that hardware. Equally, if you have got a solution to a control problem, you look for ways of approximation with the hardware available and not implementing it exactly. One looks at cost-effectiveness. So once again, it does come back to not pushing particular solutions. I think Dr. Lundh came up with a plea that we should not be DDA engineers, or general purpose computer engineers or analogue computer engineers but that really all these various techniques should be looked at in the context of the particular problem and the best one applied.

H. FRANK: Talking about DDC in computer applications normally means at the first hand to improve the values to be controlled by comparing and correcting the measured values with values computed out of other data. Often these techniques enables us to control values which are too much disturbed to base a normal control algorithm on the measured data alone. To do this data ensurement at a certain quantity of values to be controlled, we need a general purpose computer, and so we can use it for DDC, too, and have not to install special purpose environments to handle the control algorithms.

G. J. MOSHOS: On this question of the choice of a computer, I would like to see more clearly. What are the real issues between the various basic organizations? How are we to judge? I have listed a few elements here, that perhaps you could look at in the case of various computers, and then one could comment on what these would look like in the various organizations. They would be figures of accuracy, speed, reliability cost of implementing a design, and the lifetime of the application. Perhaps somebody would care to comment on these particular issues?

H. MICHAELIS: Is there not a certain convergence between the development of incremental and general purpose techniques?

B. R. GAINES: I think the answer to that one is both yes and no. When we talk of incremental techniques, in fact, there is no difference, whatsoever between the incremental technique of the DDA and the way that one solves differential equations on a general purpose computer, mathematically. If one looks at the problem of solving differential equations on the general purpose computer, you must immediately make time discrete. That means that instead of a continuous time, and very small increments, we get discontinuous time and fairly small increments. And if we make our increments such, then we are probably working with at least double length arithmetic in a general purpose computer. If our increments fit into our lower word only, then the only thing that propagates into the upper word is a carry signal. And in the DDA what we do in fact, is regard that carry signal as an incremental variable, and separate the two halves of our register. So in the DDA we have the digital equivalent of a strictly parallel integrator as one would realize it, and a DDA integrator. What we do not have in the DDA, is a parallel adder used as such. We add increments at the beginning of the integrator so when we add we must always be going into an integrator. We do not have the equivalent to a parallel multiplier and until I suppose about a year ago, it was rather stupid to think of parallel multipliers as components. I think in Richardson's book on digital systems he actually mentions this possibility of parallel multipliers, dividers, adders used as one word in an analogue computer. However, without doing too much advertizing for Texas Instruments, one of the components we used very

extensively right throughout our computers is their quad full adder. If one is working to 0.1 percent accuracy obviously three of those give you a parallel adder. If one then wants a parallel multiplier, one has just to stack something like 100 of the adders, 25 packages together, and you have got a parallel multiplier. With those components, and obviously a parallel inverter is a very easy thing to do, with the adder, the inverter which gives you the substractor, the multiplier and the integrator, which is very much like the ordinary DDA integrator, you have got a completely parallel digital analog computer.

What you use it for, I am not sure, but it is at a stage where we certainly could make it. I have never seen it put forward as a system, I do not know whether anybody made it, it is certainly in Richardson's book, and it is obviously very-very closely related to DDAs. I think in Mr. Shemer's paper he has a step towards it, in that his DDA modules, or let us say, his modules can be used as integrators in the DDA sense which he called an incremental mode, they can also be used as adders, because they are really mainly adders, and they can also be used as serial multipliers apart from the fact that they are sequential. So we have a step there towards this type of machine, but whether we will ever be at the stage where we want to apply this, as we do an analogue machine, to problem solving, is a difficult question, certainly the techniques are available and the economics are quite reasonable at present.

Г. Х. БАБИЧ: Я хочу подчеркнуть нашу точку зрения по обсуждаемому вопросу.

1) Если мы имеем простую и дешевую G. P. Computer которая может решить перед нами задачу, мы должны применять такую машину.
2) Если в системе автоматического управления мы должны экономить быстродействие, вес и т. д., мы должны экономить производительность машины. Для этой цели можно применять специализированные структуры, т. к. в противном случае мы придём к нерациональным затратам производительности машины. Мы будем иметь гибкость, но мы не сможем решить задачу в реальном времени.
3) Для некоторых классов задач известные методы дают возможность построить более экономичные структуры, но эти методы не являются единственными для специализированных машин. Мы должны комбинировать их с другими специализированными методами в дополнении к универсальному принципу.

R. A. SHEMER: I think it is generally conceded by all of us that if we are going to optimize any particular control system, we are going to have to use all the techniques available. But it seems equally conclusive that there is never going to be any particular algorithm which will tell us which is the best technique to use. It is similar to the design of normal GP computers, in that you have volumes of automata theory, but when you come to design a computer it does not tell you which is the best autocode to choose. That is obviously something at the discretion of the engineer. Eventually we may get rules for deciding, such as Mr. Babich proposed for different types of differential equations. But in between, we must try and reduce the development cost of any special purpose computer that we do decide on, because that has been one of the features encouraging GP computers. They are so flexible, they can be used for so many things, and the development cost is of course spread across the whole class of problems they are used for. And even though they are generally inefficient, in some cases, this is still the cheapest solution.

The way in which we have been looking at this problem at STL is this: We already have within ITT programs which given particular logical configurations, simulates them, lays them out on boards in an optimum manner, does the design of the printed circuit, and on the backboard does the wiring as well. And within a few years' time it is not difficult to envisage the fact that you could have this sort of thing on-line, actually manufacturing your computer. We are working in fact from the other end, in designing a compiler which will accept commands at a fairly high level, such as "rotate vector" or "multiply coordinates", so that with the systems approach of Mr. Gaines' phase-type computer or something which I spoke about, given a list of these commands we could in fact envisage a computer to do this particular thing, sort of coming out at the other end. And so development costs would be kept at the very lowest minimum. And meanwhile, as we go along, in later years we can make this formulation of the problem less and less specific, as we begin to differentiate between the different types of problem and the best modes of operation for them.

B. R. GAINES: I do not think I can sum up without answering Prof. Shearer's remarks, and questions.

It has been most useful having a member of the IFAC Components Committee along, sitting there and interjecting multiprocessor computers every now and again, causing a transient which gradually dies out, and if we look at the Proceedings afterwards, we are going to see his question and about ten remarks later an answer to it, and another question coming in.

On the multiprocessor computer configuration, we did not find the answer to the question as to what are the advantages, or whether there were operating systems, which could legitimately be said to have advantages. I think if one looks at the extent to which special purpose functions have been implemented in the processor, on the early machines we really only did have add, complement and a few logical operations. On the most sophisticated machines at present obviously there is multiplication and decision, but both are in floating point form and not beyond this. I think the reasons for this, if one really wants to state them rigorously, come down to some mathematics, and hospitality has been so good in Hungary I can not get my mind around it. But the more one does with the processor, the more messages one needs to the processor to tell it what to do. If we take a particular case — the present processor, with an accumulator, in fact operates on two numerical or logical variables, and produces a single result. There are not many operations you can do on two variables: ad-

dition, substraction, multiplication, square root, cube root, there is a variety of possible operations, but the ones that one actually wants in practice are very few. Now as soon as you go to three variables, in instructing the processor you now have to tell it where these variables are, or you have got to load them into the processor to start with, and just let it go on proceeding, working with those variables. Now, as soon as one starts to have to tell the processor where those variables are, you immediately increase the length of the word instructing the processor to a tremendous amount, and we know that getting words out of memory, in fact, is what takes the most time. This is one of the defects of going to a more specialized processor with extra operations. And in practice it does seem that the real limit gets to be towards floating point arithmetic.

There are possibilities if having fast scratch-pad memories, loading them up with a reasonable amount of data, and then doing extensive computation on them. For any particular applications and air traffic controllers is always one that springs to mind, because very large computer systems (and multiprocessor ones) have been used quite extensively in ATC, these are specialist applications. I am certain ATC will have quite a large market, but it does not solve the general problem. When one is talking about software, there is obviously software at several levels. One of the things is just having the storage capability for holding all the instructions, knowing that quite a lot of the very long instructions one is not going to be using very often. We are now working with completely variable length words as far as data are concerned with very flexible data formats, and we can equally go to very flexible instruction formats. But the returns from this do not seem worthwhile at present. Yet it certainly is not a black and white case. You can not just put six processors in a computer and work almost six times faster. If you put six processors in, and if you are not very careful, you will be working slower, because you have got to have more instructions. And one of the big defects, of course in a multiprocessor computer, where you have got several programs running concurrently, is that the programs have certainly got to be performed procedurally. You can not load up your processor with data, and assume it is going to be there. Every time another program takes up the processor, you have got to dump all the data that is in the processor, and start again. And your subroutines are whithin core, and must have been changed by particular programs. It obviously is a very complex area, and there may well be a tendency to use a lot more medium sized machines rather than to go for very great central facilities. This is an area of difficulty.

As far as the DDA is concerned, obviously this is a multiprocessing machine. If one looks at it from the point of view of the solution of differential equations, they satisfy the requirement of being very special purpose, and it is ideally suited to those particular equations. The other points were, what features are we looking for: accuracy, speed, reliability and so on. One of the important advantages of the analogue computer is that analogue, of course, is a word with a meaning in its own right, with engineering analogues for one thing and another. As far as differential equations are concerned, the analogue computer is called such, because it forms an analogue of differential equations. When we go to other problems, what we are really looking for, if you want a computer which is easy to use, is a computer which is an analogue of the operations which we have got in our problem set. Now if our problem set has factors of speed coming in, and factors of extreme reliability, then these are obviously going to have to be built into our computer. If we actually want to know whether a computer is reliable, we have got to have checks for it, if we want not only to know when it has failed but also to pull in other computational facilities to stop it failing effectively, then we have obviously got to have multiprocessing hardware. What we tend to do, is to think of a specific problem, and not to think of all these surrounding factors as being part of the problem itself. Solving differential equations is an entirely different problem, for example, if it is in real time or we just want a numerical solution completely off-line. To conclude, I think that all our thanks are due to Dr. Hatvany who has obviously been the influence behind this whole discussion, and the whole Conference. I think it has given an opportunity for a lot of material which otherwise might not have come into the literature or would have been buried in specialized literature, to become available to control engineers through the Proceedings, and for the discussion of the various problems, also to thresh out some points which are somewhat clearer now than when we first gathered together. I think the really important point is that within various crews working on control problems, given a particular problem, incremental techniques do crop up as a possible solution. Now whether or not the techniques in themselves are going to have a major impact or not, is a function of other factors, of education, economics, and the direction of control engineering itself. But there is no doubt at all, that this material should be made available in some way, otherwise we are always going to be in the position, as I think several of us have been, of rediscovering things which have been discovered previously. I think just on that basis the Conference and the discussion itself has been very successful.

J. HATVANY: It only remains to me to thank all participants in the discussion in particular to thank the authors of the papers and most particularly to thank Dr. Gaines for his excellent leadership of the discussion. I must, however, say one word in response to what he addressed to me, and that is that it is of course we, in Hungary, in particular in our Institute, who are a group working on incremental techniques and who have been very busy for some years rediscovering all that other people have done, who should be grateful, and are indeed very grateful for receiving all the extremely valuable advice, hints and views which we have received, and hope to receive in the coming days in the course of this Symposium. Thank you very much.