5 L

CLM-P281

United Kingdom Atomic Energy Authority
RESEARCH GROUP
Preprint

COMPUTERS AND PHYSICS

K V ROBERTS

Culham Laboratory
Abingdon Berkshire

1971

CLM-P 281

COMPUTERS AND PHYSICS

by

K.V. Roberts

(Paper presented at the IAEA International Centre for Theoretical Physics, Seminar Course on Computing as a Language of Physics, August 1971)

ABSTRACT

This introductory lecture begins by discussing from a fairly fundamental point of view what computers really do and why they should be important to physics. For example, how significant has their impact been in the quarter century which has elapsed since the electronic digital computer was invented, and what may be expected of them in the future? How can we ensure that they realize their true scientific potential and that massive programming effort is used to maximum effect? Does Computational Physics have something to contribute to Computer Science and Software Engineering? A brief look is then taken at one particular field of computational physics, namely the numerical solution of sets of coupled partial differential equations which describe the time evolution of classical systems.

U.K.A.E.A. Research Group,
Culham Laboratory,
Abingdon,
Berks.

August 1971

1. INTRODUCTION

I shall begin this introductory lecture by discussing from a fairly fundamental point of view what computers really do and why they should be important to physics. How significant has their impact been in the quarter century which has elapsed since the electronic digital computer was invented, and what may be expected of them in the future? How can we ensure that they realize their true scientific potential and that massive programming effort is used to maximum effect? Does Computational Physics have something to contribute to Computer Science and Software Engineering? I shall then take a brief look at one particular field of computational physics, namely the numerical solution of sets of coupled partial differential equations which describe the time evolution of classical systems, in preparation for some of the lectures which follow.

An excellent review of the subject is given in the recent book 'Computers and their Role in the Physical Sciences', edited by Fernbach and Taub (1970) This describes the origin of the electronic digital computer (in which computational physics played a considerable part) and gives many references. More specialized papers can be found in Journal of Computational Physics (Academic Press), Computer Physics Communications (North-Holland) and the annual review series Methods in Computational Physics (Academic Press). The International Physics Program Library, operated by Queen's University, Belfast in association with Computer Physics Communications, has recently been established to publish the programs themselves in digital form.

Figure 1 indicates the main branches of computational physics, together with certain fields which might more properly be regarded as part of computer science (languages and translators) or software engineering (operating systems). The relation between these fields and computational physics may be compared to the relation between mathematics and theoretical physics, or between engineering and experimental physics (Figure 2). Good languages and good operating systems are vital to the proper growth of computational physics and therefore physicists can be expected to play a part in their development just as they have always done in many branches of mathematics and engineering.

Figure 3 represents the interplay between the three main ways of approaching a physical problem; experimental, theoretical and computational. Each has its characteristic methods of approach, its advantages and limitations, some of which will now be mentioned.

Theoretical Physics

Theoretical physics makes considerable use of analogies, many of which are geometrical in character; for example the calculus was originally based on the idea of gradients and areas. Familiar concepts in three dimensions are generalized to n or to an infinite number of dimensions. It relies heavily on the use of symbolism, enabling many actual cases to be described by a single algebraic formula. Theoretical physics is position-free, since it can survey any portion of space-time with equal ease, for example the inside of a neutron star at some distant epoch. And it is scale-free, ranging at will from the scale of a quark to that of the whole universe, and from 10^{-23} seconds to 10^{10} years. It is universal, in the sense that one piece of theory, such as Coulomb's law or Laplace's equation, can be applied to innumerable actual situations.

Theory makes extensive use of linearization. There is almost a motto, "When in doubt, linearize". Any linear process is relatively easy to solve by analytic techniques, and weakly non-linear processes by perturbation theory. Strongly non-linear processes are much more difficult. Symmetry and conservation laws are related to one another and play an essential role, not only in basic theory but also in the solution of practical problems, as by the method of separation of variables. Complex function theory has a similar dual role; it appears to be fundamental to high-energy physics and to the theory of ordinary differential equations and at the same time is of great practical use in the solution of 2-dimensional problems because of its relation to Laplace's equation. Many of the mathematical methods used in theoretical physics have been summarized by Morse and Feshbach (1953).

Approximation techniques are essential. Sometimes this means separating out a few of the many degrees of freedom of a large system, or distinguishing between widely different time-scales as in the method of adiabatic invariants. In other cases such as statistical mechanics the number of degrees of freedom is treated as infinite since this makes the formulae much simpler.

These are some of the mathematical tools; practical tools include pencil and paper, chalk and blackboard, books, journals and meetings. Theoretical physics is cheap but it requires high IQ. Another important feature is that theory is self-enhancing; by practising it one becomes a better theoretician. This is not necessarily true of experimental physics (which requires the organization of staff and finance, the building of apparatus and the management of contracts), nor of computational physics (which involves struggling with awkward and unreliable computing systems, much handling of cards and paper, and a continual search for errors in the programs). A major task will be to build this feature of 'self-enhancement' into computational physics by improving the techniques.

Experimental Physics

Experimental Physics provides the ultimate test and source of information for theory, just as theory provides the equations to be solved by computation. With great ingenuity the scope of experiments and observations has gradually extended both ways from human scale to the range 10^{-13} cm - 10^{10} light years in length, and 10^{-23} secs - 10^{10} years in time. But experimental physics is neither position-free nor scale-free, and the cost of an experiment depends very much on the scale of the phenomenon which is under investigation. Where the expense is high it may be preferable to use theory or computation, although experimental modelling is often also of great use.

No experiment is exact and the potential sources of error must always be carefully examined.

Computational Physics

Computational Physics combines some of the features of both theory and experiment. Like theoretical physics it is position-free and scale-free, and it can survey phenomena in phase-space just as easily as real space. It is symbolic in the sense that a program, like an algebraic formula, can handle any number of actual calculations, but each individual calculation is more nearly analogous to a single experiment or observation and provides only numerical or graphical results.

To some extent it is possible to solve the equations on a computer without understanding them just as one can carry out an exploratory experiment. With more complicated phenomena involving a considerable range of length and time scales it is however essential to make analytic approximations before putting the problem on to the computer otherwise impossibly large amounts of machine time or storage space may be needed. Not more than about 10^6 degrees of freedom can be handled on present-day computers, or 10^3 if they all interact with one another. Thus computational physics can fill in the range between few-particle dynamics and statistical mechanics.

Diagnostic measurements are relatively easy compared to their counterparts in experiments. This enables one to obtain many-particle correlations, for example, which can be checked against theory. On the other hand there must be a constant search for 'computational errors' introduced by finite mesh sizes, finite timesteps etc., and it is preferable to think of a large-scale calculation as a numerical experiment, with the program as the apparatus, and to employ all the methodology which has previously been established for real experiments. (Notebooks, control experiments, error estimates and so on).

Computational physics is particularly suitable for non-linear, non-symmetrical phenomena where the usual theoretical methods do not apply (such as in weather calculations), but often the programs are easier to write and the calculations go much faster in simple situations such as rectangular Cartesian geometry with rigid perfectly conducting walls.

It is often possible to take situations that normally are only handled algebraically and to display them in pictorial form. Thus computing can put life into somewhat abstract subjects and might be of great help, for example, in the teaching of complex variable theory.

Finally, there is great danger if computational physicists become too preoccupied with mundane details of computing at the expense of the physics itself, but the only solution here seems to be to get the details right once for all, just as at one stage it was necessary to introduce rigorous limiting processes into mathematics.

Examples

Sometimes one method of approach will be more appropriate and sometimes another; frequently they will work in pairs and at times all three methods must be used together. An example where computational techniques are particularly appropriate is in the solution of equations which describe the internal structure and evolution of stars (Iben, 1970). The equations are complicated and non-linear but they are well-defined, and provided that spherical symmetry can be assumed they are well within the range that computers are able to handle. On the other hand analytic methods have difficulty because of the non-linearity, while it is clearly awkward to do experiments or even to make observations (except with neutrinos) in the interior of a star.

The book by Betchov and Criminale (1967) on The Stability of Parallel Flows gives a good account of the way in which analytic and computational techniques can support one another in one branch of fluid dynamics. Harlow (1970) has provided a general bibliography of papers dealing with numerical techniques for solving 2- and 3-dimensional time-dependent problems in fluid flow.

Physics and Information

The purpose of a computer is to process information as we shall see in §2. Now physicists do spend a great deal of time handling information of one kind or another and any impact that computers have on physics must eventually result from this fact. Many of the techniques used for handling scientific information have reached a high degree of sophistication, particularly in theoretical physics, and here it is likely to take a long time before computers can compete on equal terms; for example the developments in the physical sciences which occurred within 5 years due to the discovery of Schrodinger's equation can hardly be paralleled by those which have occurred within 25 years due to the invention of the electronic digital computer. But in cases where conditions have been more suitable for the introduction of computers, such as the processing of large amounts of digital data from measuring devices and the automatic control of experimental equipment, their impact has been more obvious.

2. HARDWARE

Let us therefore go right back to the beginning and try to see what computers can in principle do. Basically, an electronic computer is a device for handling binary information or data contained in a fast memory or store. The data is conventionally represented as an ordered set of 0's and 1's (bits), grouped into bytes and words. In processing this data the computer obeys a sequence of instructions which are themselves represented by binary information and are drawn from the same store (Figure 4). The sequence of instructions is called a program.

It is preferable to think of the program as fixed information, while the data will in general vary during the course of the computation. There is in fact an interesting analogy between a data processor and a dynamical system, in which the program corresponds to the Hamiltonian $H(q,p)$ while the data values correspond to the complete set of canonical coordinates and momenta $\{q,p\}$ which between them define the current state of the system. The progressive modification of the data by the program as the computation proceeds then corresponds to the time evolution of the dynamical system.

The data values can be made to control:

(a) The action of the current instruction.
(b) The location of the next instruction to be obeyed.

This facility enables the program to make decisions which depend on the current data values, and is nowadays usually implemented by first transferring the necessary control information from the main store into subsidiary fast storage devices called registers, one or more of which can be consulted while an instruction is being interpreted.

Dynamic Program Modification

Because the program instructions can also be regarded as data it is possible, in principle, for a computer to process its own instructions during the course of a run. This is quite a fundamental idea because it means that the program itself can evolve dynamically, as well as the data values. At one time this property was regarded as essential (Goldstine and Von Neumann, 1963; Elgot and Robinson, 1964; Goldstine, 1970), but it seems that the essential tasks have now been taken over by the use of registers, and self-modifying programs are currently regarded as bad practice because they are so difficult to understand. No legal Fortran or Algol program can modify itself for example.

In mathematics one sometimes finds that a generalization is remarkably productive and leads to a host of new results (real → complex numbers); at other times it almost seems to kill progress altogether (time-dependent Hamiltonians $H\ (q,p,t)$ or non-Hamiltonian systems; groups → semi-groups). We do not know on which side of the fence these dynamically, self-evolving programs are likely to lie. I shall not discuss them further here but it may be that this is an area where substantial advances in computational physics will one day be made.

The Universality of Hardware

There is a sense in which all computers are the same (Pasta, 1970) :

"As an example of the kind of thing we are talking about, consider the Turing machine, a model invented in the 1930's by the mathematician A M Turing*. This abstract model is very simple. In one form it is a device with a finite number of internal states and a tape of arbitrary length marked into squares. At any moment it can read a symbol on the tape. Based on that symbol and on the internal state, the machine can initiate actions to change the symbol and to move the tape one square left or right.

One would expect such a machine to be limited in the kinds of things it could do and yet Turing showed that any effective computation performed on any computer can be performed on a Turing machine. The universality of this machine allows us to establish truths about it which will apply to all other machines and consideration of this and other equivalent models has increased our understanding of computers, programs and computations, all of which can be fit into this simple model".

Turing's theorem suggests that any fundamental advances in computational physics are much more likely to come from better theoretical techniques, from improved algorithms and languages or from software engineering than from improved hardware. During the past 25 years there have been steady quantitative improvements in the architecture of computers and in their speed, storage size, reliability, versatility and convenience, together with a parallel decrease in the cost per unit of computation, but there have been no radical changes of principle.

Some Practical Improvements

There are however a number of potential improvements of a practical kind whose combined effect might be so dramatic as to appear fundamental. These include:

(a) Networks of computers linked together via the communications system.

(b) Massive direct-access storage devices.

(c) Ultra-high-speed character and vector displays.

(d) An extended character set, including the Greek alphabet and mathematical symbols.

(e) Improved ergonomics of man-machine interaction.

(f) Further decreases in cost, and improvements in reliability of on-line systems.

These developments might make it practicable for a 'power-assisted' algebra facility to be introduced, by which a theoretical physicist working at a console could automatically and almost instantaneously manipulate analytic expressions appearing on the screen by issuing commands to the system to perform standard transformations and integrals. This has already been partly implemented at Stanford University in an experiment on the teaching of elementary algebra in schools, but in order to compete with pencil and paper it is important to get the practical details right.

Very fast, powerful and selective information-retrieval facilities might also become possible, enabling a scientist working in one field to familiarize himself rapidly with the state-of-the-art in another. In this connection, a fundamental technique that has been developed in computer science might well be applied to reduce the bulk of the regular scientific literature, namely that of the subroutine or macro. Theorems, diagrams, formulae, definitions, conventions etc. which are constantly being reproduced in full could be stored in one place and automatically called into use when required simply by naming them. At the same time, the notation could be automatically changed to fit that of the paper in which they were called.

Another possibility is to have a dynamic style of publication, containing not only algebraic formulae but programs for evaluating them numerically or displaying them graphically on a screen as a function of parameters selected by the 'reader'.

3. DATA TRANSFORMATIONS

So far we have only considered binary strings of 0's and 1's. These are not in themselves very interesting and their importance lies in the ease with which they can be transferred to and from other types of data format (Figure 5). Binary or 'digital' data is freely interchangeable between electrical signals, magnetic recording media and holes in punched cards or paper tape although at different speeds. Electrical information can readily be converted from analogue to digital form and vice versa although with some loss of content. Apart from this, it should be emphasized that output by the computer is usually much faster, cheaper and more convenient than input as illustrated by the dashed lines in Figure 5. It is relatively easy for a computer to display a table, draw a graph or make a movie film or even to talk, but much harder to get this information back into digital form. Therefore so far as computers are concerned digital information ought to be regarded as the primary form, while printed output, graphs, speech etc. are temporary forms intended only for communication with people.

Digital Information

Digital information has a number of important advantages. It can be transmitted almost instantaneously from point to point, updated, duplicated, stored and retrieved, automatically manipulated in different ways and displayed to people in a variety of forms. We can in fact regard a set of data as an operand D and a display program as an operator P_i , various forms of display Δ_i being generated as products

$$\Delta_i = P_i D \tag{1}$$

If for example a calculation leaves its output in a random access file, then not only can a physicist working at a console cause the results of the calculation to be displayed in various ways so that he can understand their meaning, but he can also use the same file as input for a further series of calculations. These advantages are lost if the output file is simply printed and then destroyed.

Digital information does however have a number of grave disadvantages which must be carefully taken into account if it is to serve as a medium for scientific communication. It is extremely fragile, and on many computer systems even minor damage to an index can cause all the data on a storage device to be lost. Few of the scientific discoveries of antiquity would have survived if their recording media had suffered from this disability. And digital information does rely heavily on good indexing; compare browsing through a magnetic disc file with browsing through a library of scientific books.

4. ALGORITHMS, PROGRAMS AND SOFTWARE

Computers can carry out any process which we know how to reduce to algorithmic form; that is any process for which we can prescribe a definite set of rules no matter how complicated. Ultimately this process must be reduced to the manipulation of a binary bit pattern and the algorithm itself must be expressed in a similar form (Figure 4), but in practice we can develop our algorithms in a more convenient language and then use a second algorithm to carry out the conversion automatically (Figure 6). In fact a primary input device such as a teletype usually performs a preliminary conversion to binary form, and this is then subsequently transferred by one or more systems programs such as compilers, link editors etc. until the binary instruction code of the machine is finally reached.

Three requirements are:

A. It must be possible to find an algorithm to carry out the required process.

B. The algorithm must be coded for a specific machine.

C. The number of computer operations required for the process must not be too large.

Much of the effort in computational physics at the present time is occupied by B, and since this is rather a mechanical task it tends to divert attention from the physics proper. However just because it is a mechanical task it should itself be automated. The ultimate solution is one in which the languages which are most suitable for people who are investigating and expressing the algorithms are also intelligible to computers, and can be automatically converted by them into efficient binary code. Some comments on how this may be achieved will be made in §8, in connection with Symbolic Algol.

Algorithms for some of the processes used in physics have existed for many years, for example, arithmetic, and the solution of sets of coupled ordinary differential equations by finite difference methods. Here the computer was able to make an immediate impact. In high energy physics a great deal of effort has been put into algorithms for pattern recognition in connection with the processing of bubble chamber data, and with considerable success (Snyder, 1970; Kowarski, 1970). Some success has been achieved with automatic theorem proving and with the automatic solution of elementary integrals by analytic methods, but neither have influenced physics as yet. In other cases where theoretical physicists have no algorithms and must proceed intuitively, as in the formulation of new concepts, computers have also naturally had little influence.

Algorithmic and Programming Languages

In order to satisfy A and B it will be necessary to develop

D. Powerful, intelligible algorithmic languages.

E. A substantial body of algorithms expressed in these languages, for the solution of physical problems.

F. Means for converting E into efficient binary machine code for the various types of computer system.

A high-level programming language such as Fortran or Algol enables algorithms to be expressed in a form which is relatively convenient for people to use, while at the same time allowing them to be translated without too much difficulty into reasonably efficient machine code. Algorithms are in many ways similar to mathematical theorems, and need to be made intelligible and universal for the same reasons. Unfortunately the existing languages cannot be compared in scope to mathematical notations such as non-commutative algebra and the tensor calculus. Furthermore it is nowadays very difficult to introduce a new programming language because of the cost of developing and maintaining the necessary translators or compilers for a variety of different computer systems. The result has been that for physicists the state-of-the-art has remained frozen for many years; although many research languages have been developed by individual computer scientists during the last two decades only Fortran (introduced in 1957) and Algol (introduced in 1960) are of major importance in physics. These have awkward deficiencies which in principle could be easily put right, but which remain uncorrected because of the difficulty of reaching international agreement and then modifying all the existing compilers. The restriction to 6-character identifiers in Fortran and the omission of complex numbers, COMMON and EQUIVALENCE declarations and standard input-output facilities from Algol are typical examples.

It was hard for mathematics to progress until a good notation had been introduced in order to express the operations of arithmetic and algebra (Ball, 1908); try calculating in Roman numerals! It has also been said that the development of English mathematics was held up for more than a century by reliance on the methods and notation of Newton rather than those of Leibnitz. Computational physics is likely to remain equally constrained until it becomes a straightforward matter to introduce powerful new notations in which algorithms can be expressed. Even the hardware restriction to upper case letters, numerals and a few special characters constitutes a severe limitation, compared to the great variety of symbol types, sizes and positions which are exploited in mathematics.

The solution appears to be for scientists themselves to develop and publish machine-independent or portable compilers, program generators and macro-processors in addition to the growing literature of applications programs and packages. If these are written in modular form and well documented it should be relatively straightforward to extend them to meet new situations in the same way that mathematical theorems are continually generalized and extended.

5. A SCIENTIFIC SOFTWARE LITERATURE

Clearly theoretical physics would hardly progress at all if every worker had to build up all the mathematics that he needed right from the beginning, and it will not be practicable to develop the enormously complex algorithms and programs that will be required in computational physics unless each individual is able to stand on the shoulders of his predecessors in a similar way.

The Coding Problem

It might be argued that although the algorithms themselves should be published in the regular scientific journals, coding them for a specific application should be left to the individual worker. This however is unrealistic because of the very high cost of coding and because of the long delays involved.

Some figures have recently been published on the costs of computer software and the effort needed to write it. For IBM 360 software the cost of <u>each instruction</u> has been estimated at $50-60, with 0.2 instructions produced per man-hour (Bemer, 1970). Figure 7 shows the growth in software requirements in terms of lines of code for successive machines (McClure, 1969), while Figure 8 expresses it in terms of millions of man-hours spent (Bemer, 1970). Both increase exponentially with time, by a factor of about 200 in 10 years, and it seems that both Parkinson's law and the Peter Principle must surely be in operation (David, 1969). Bemer remarks:

> "My nightmares come from imagining a new system scheduled for 1972. If the McClure chart holds true to give 25 million instructions, then the best figures we have say it will cost a billion and a quarter dollars, produced by 15,000 programmers."

Yet according to Barbe (1970) only 2% of the $36 billions' worth of software in operation in the USA is transferable from one computer to another; the rest is doomed to die with the hardware.

Physicists may be doing a little better, since Snyder (1970) estimates that a 60,000-word bubble chamber analysis program written in Fortran might require 10 man-years of programming effort which would represent a coding rate some 15 times faster.

<u>Publication, Portability and Modularity</u>

Since computational physicists do not generally have this amount of money to spend, the operating systems, compilers and applications programs which they need will not get written unless some better method is found. It does appear however that three techniques which have worked well in science and mathematics in the past could go a long way towards solving the problem.

The first technique is that of <u>open publication</u>. The new journal Computer Physics Communications (North-Holland) has recently been founded to publish details of well-documented, refereed, tested physics programs. Associated with this is the International Physics Program Library at Queen's University, Belfast which publishes the programs themselves in digital form. I have argued elsewhere the advantages of such a scheme (Roberts, 1969). One important advantage is that by 'exposing' the program listings to public criticism the standards are likely to be forced up. A primary reason why the standards in software engineering are so low is that there are so few models to work from, because programs are regarded as commercially valuable and are not therefore seen by more than a few people.

The existence of a high-quality open scientific <u>program</u> literature should serve as a stimulus to the whole computing industry, just as the regular scientific and mathematical literature of books and journals does for technology.

The next technique is that of <u>portability</u>, which is the same as 'universality' in science and mathematics. Once a new program, subroutine package, compiler or scientific operating system has been developed and published, it should be possible to run it at any scientific laboratory or university throughout the world, just as one can read any journal article or textbook. There are two basic requirements for this:

(a) Scientific libraries must be persuaded to subscribe to the journal tapes, in the same way that they do to the regular scientific journals, and to make their contents as readily available as are books and papers.

(b) The published programs must be written in universally available languages.

At present only the universal high-level languages Fortran and Algol are accepted by Computer Physics Communications. An important further requirement is a lower-level universal language in which compilers can be written, and in which they can generate their output (Figure 9). As soon as this is available, a <u>single</u> implementation of each new language will make it available on <u>all</u> machines, thus saving both excessive duplication of effort and also the dangers of different implementations being out of step as happens with Fortran and Algol at the present time.

This idea was proposed many years ago, in connection with the so-called Universal Computer Oriented Language or UNCOL (Mock <u>et al</u>, 1958). It enables N languages to be implemented on M machines with a total amount of effort N+M instead of NM. Another possible way of implementing the idea is by means of macro-processors (Poole and Waite, 1970). It seems unlikely that new scientific languages can be universally introduced except in some such way as this.

Figure 10 illustrates what I believe the structure of the <u>scientific software literature</u> Σ should eventually be; it has been drawn to parallel Figure 1 which shows the structure of computational physics itself. Note that it <u>includes</u> the regular scientific literature, since I have assumed that in due course books and journals (or at least automatic indexes to them) will be made available in digital form. There is a significant danger here. In the past the scientific literature has always been completely 'visible' even though it has been published, in large part, by commercial firms. If it ever gets transferred to proprietary data banks which can be automatically consulted, for a fee, but can never be openly inspected by the scientific community as a whole, then there is a great danger that it will become corrupted. Even the standard Fortran library functions often contain mistakes.

Once established, Σ may be expected to increase steadily with time like the regular scientific literature and to be equally permanent; there are already programs that have been in use for more than 10 years and which have been run on a whole series of machines. At any given epoch, Σ will be run on a variety of different hardware types H_1, H_2, H_3 .... As Σ expands, it will be less and less economically practicable to recode even major portions of it for each new hardware system H_α, and this is why portability is essential. The most that can be done will be to recode certain <u>replacement modules</u> R_1, R_2, R_3. .... (Figure 10) which are executed with very high frequency and so occupy a substantial fraction of the computer time.

The third important technique is <u>modularity</u>, which is the same as the 'Principle of Abstraction' in mathematics. Theoretical Physics works by developing a number of separate tools, e.g. vector algebra, tensor calculus, group theory, Green's functions, Laplace's equation, and then combining them together in many different ways. This means that when a new branch of theoretical physics has to be mapped out much of the necessary mathematics is already available (as with Schrodinger's equation and quantum mechanics in 1926). It also means that theoreticians can often move freely from one field to another because they recognize the language.

Suppose that we build a set of program modules of n different types, with m modules of each type. Then by combining these together in all possible ways the number of complete programs we can form is of order m^n, while the work required is only of order mn. Even allowing for many non-viable combinations this is still a considerable advantage. To put it in another way, suppose that a single new module is developed of one particular type; then m^{n-1} new programs can in principle be constructed from it, an <u>amplification factor</u> of Nm^{n-1}.

Typical examples might be the introduction of a new type of coordinate system (e.g. spherical polars), or a new graphical display package. Provided that the existing programs are properly constructed, many of them can quickly make use of these with little further effort.

6. THE POWER AND LIMITATIONS OF COMPUTERS

I have stressed the organizational problem at some length because this is the single most important practical task facing computational physics at the present time. There are many algorithms in the literature which are not being exploited because of the effort needed to code them. There are many good programs that can only be used in one or two major laboratories (notably the Los Alamos hydrodynamics codes), and others which have gone out of use because their originators moved on to other work. Also there are large numbers of significant research languages which have not moved very far from the computer science departments where they were developed; meanwhile, Fortran has been frozen since 1964. However, these are all problems which can be solved by persuasion and good planning, along the lines I have already indicated. A more basic problem is whether or not there are any fundamental limitations on the use of computers in physics.

It is sometimes thought that computers will eventually kill theoretical physics; all that one will need to do is to program the equations and press the button in order to get a numerical answer. This is very far from being the case. Consider an assembly of N particles, interacting via Newton's laws of motion and gravitation. If N is small (say equal to the number of planets together with the sun), then it is indeed possible to solve the equations rather accurately over long epochs using the computer, and in this sense one might say that much of the analytic work done in the 18th and 19th centuries on the classical few-body problem in astronomy was not strictly necessary. Fortunately computers were not available then because the modules developed during the course of this work (e.g. Lagrangian and Hamiltonian mechanics and perturbation theory) turned out to be of great use in other fields such as quantum mechanics and statistical mechanics.

Because the number of elementary interactions between N particles increases as N^2, straightforward computational techniques become impracticable as soon as the number of particles greatly exceeds 100 or at most, 1000. Statistical mechanics is difficult to apply because of the infinite potential energy that can be released when two gravitating particles approach each other, and the two lines of attack, theoretical and computational, must support one another.

Theoretical physics relies to a large extent on finding adequate approximations. Often this is a question of separating the various timescales in a problem. For example the Born-Oppenheimer method used in molecular theory treats the nuclei as fixed when calculating the electron energies and wave-functions, from which one obtains a potential function to be used in solving the motion of the nuclei themselves. Timescales are equally important in computational physics because if a naive approach is adopted, the cost of the calculation will be determined by the shortest timescale t_{min} of the problem and will rise to astronomical values if the ratio of this to the largest timescale t_{max} becomes too great.

In the case of an assembly of N gravitating stars the shortest timescale is likely to be determined by the orbital periods of close binaries which can decrease without limit. These must somehow be decoupled from the problem, e.g. by treating the motion analytically until the perturbations due to nearby stars become too great. The N^2 difficulty might be removed by replacing the effect of the interactions outside a given distance by that of a mean field, so that the amount of calculation increases only as N.

Research of this type often proceeds in one of two ways:

(a) Theoretical approximations are devised to remove difficulties encountered in the computation, and then these approximations are verified using the computer.

(b) The numerical calculations turn up unexpected and striking results, which can then be given a simple analytic explanation.

Thus the theoretical and computational approaches are complementary to one another.

One instance where computers could have been of great assistance during the 19th century is in the solution of the Navier-Stokes equations for viscous flow. If these equations had been solved numerically in 2 dimensions at moderate Reynolds numbers, boundary layers of finite thickness would have automatically developed in the neighbourhood of solid surfaces, and the interpretation of this phenomenon should have led to the discovery of boundary layer theory and an understanding of the problem of flight much earlier than actually occurred. Shocks and Karman vortex streets would have automatically turned up in a similar way.

It is interesting to notice another complementarity between the theoretical and computational approaches, since theory finds it easier to deal with _thin_ boundary layers, while computers find it easier to deal with _thick_ ones (covering several space steps).

Partial Differential Equations

When we turn to partial differential equations the limitations of computers become even more apparent. Excluding high-energy physics for which the equations themselves are not well-defined but their number seems to be infinite, we find three situations in decreasing order of complexity as indicated in the table.

Numbers of Dimensions

Schrodinger	Configuration Space	3N
Vlasov	Classical Phase Space	6
Navier-Stokes	Real Space	3

Vlasov's equation describing the phase space motion of particles interacting via long-range fields is important in plasma physics, while the Navier-Stokes equations of hydrodynamics are a prototype for many similar sets of coupled partial differential equations in magnetohydrodynamics, astrophysics, geophysics and other fields.

Assuming that we need at least 100 space points in each direction to achieve good accuracy (i.e 25 Fourier modes with $\geqslant 4$ points/mode), the amount of storage needed is 100^{3N} for Schrodinger's equation, 100^{6} for Vlasov's equation, and 100^{3} for hydrodynamics.

It is now just becoming practicable to compute with 10^{6} mesh points using machines such as the CDC STAR-100 so that 3-dimensional hydrodynamics problems should shortly be fairly routine provided that the Reynolds number is not too high. To achieve the same accuracy with Vlasov's equation requires a further factor of 10^{6} in storage capacity and speed which is difficult to envisage at the present time, although a factor 10^{3} can perhaps be anticipated. But this method of solving Schrodinger's equation is out of the question for all but the simplest situations.

One is again led to the need for making adequate approximations before putting a problem on to the computer and this of course is done in quantum mechanics, for example by the method of molecular orbitals (Clementi, 1970). In general, insight is likely to come not only from the numerical results themselves but also from studying the accuracy of the various approximations and trying to understand why they work as they do.

The Turbulence Problem

It has recently been pointed out by Emmons (1970) that a straightforward numerical attack on the problem of hydrodynamic turbulence in 3 dimensions is doomed to failure, since to solve the simplest turbulent pipe flow problem would require 10^{10} mesh points and 10^{14} operations altogether for a Reynolds number $R_e = 5 \times 10^3$, occupying perhaps 100 years on existing computers (or 10^{22} operations and the full age of the universe at $R_e = 10^7$). Here again one must look for a combination of more subtle computational techniques combined with physical insight and good theoretical approximations.

7. DISPERSION RELATIONS

When a partial differential equation is solved on a computer one effect is to change the dispersion relations of linearized perturbations, or small-amplitude waves. This happens because derivatives are replaced by differences, so that for example

$$df/dx \rightarrow \frac{f(x+\Delta x) - f(x-\Delta x)}{2\Delta x} \tag{2}$$

$$d^2f/dx^2 \rightarrow \frac{f(x+\Delta x) - 2f(x) + f(x-\Delta x)}{(\Delta x)^2} \tag{3}$$

The result is that algebraic dispersion relations are replaced by more complex trigonometric ones, since

$$k \rightarrow \frac{\sin k\Delta x}{\Delta x} \tag{4}$$

$$k^2 \rightarrow \frac{2(\cos k\Delta x - 1)}{(\Delta x)^2} \tag{5}$$

and so on.

Depending on the difference scheme, on the equations themselves and on the ratio of the 'mesh speed' $\Delta x/\Delta t$ to the various characteristic speeds of the problem (where Δt is the timestep), this replacement can cause stable waves to become unstable or damped, and non-dispersive waves (e.g. sound-waves) to become dispersive. A good part of the discussion at this Seminar Course will be concerned with such problems, the situation being quite analogous to the replacement of a continuum by a discrete lattice in solid state physics.

This analogy with solid state physics might usefully be exploited further. In particular since there is a maximum wave-number k_{max} that can be represented on a lattice with finite spacing Δx, when two waves $\underline{k}_1$, $\underline{k}_2$ interact to give a new wave $\underline{k} = \underline{k}_1 + \underline{k}_2$ with $|k| > k_{max}$, this energy must be diverted to some other mode $|k'| < k_{max}$ by a type of Umklapp process which is known in computational physics as 'aliasing'. This leads to errors in turbulence investigations, and the energy at high wave-numbers must be removed by some form of artificial damping before it can cause damage.

The simplest example of numerical dispersion occurs in the solution of the 1-dimensional advective equation (Roberts and Weiss, 1966)

$$\frac{\partial f}{\partial t} + v \frac{\partial f}{\partial x} = 0 \tag{6}$$

where v is a constant. This evidently describes a wave moving with uniform velocity v, thus preserving its shape unchanged, and the dispersion relation is

$$\omega = kv \tag{7}$$

Making the replacement (4) but keeping Δt small we find

$$\omega = \left(\frac{\sin k\Delta x}{k\Delta x}\right) kv \tag{8}$$

This means that disturbances of short wavelength propagate more slowly and that for

$$k\Delta x = \pi \tag{9}$$

there is no propagation at all. A pulse can leave a train of waves behind it which may be misinterpreted as a real physical phenomenon, and a function (such as the density or temperature), which according to the differential equations must remain everywhere positive, can take negative values in the numerical calculation.

Equation (6) is significant because it is the prototype of all the hydrodynamic equations, in which the left-hand side occurs as the Eulerian derivative. It is also closely associated with Vlasov's equation. Advective errors are of importance in meteorology where v represents the speed with which disturbances are carried by the wind.

8. SYMBOLIC PROGRAMMING

I discussed earlier the possibility of finding a generalized language which would be suitable not only for the formulation and discussion of algorithms, but also for programming the computer itself. I should like to finish this lecture by mentioning how this has been very largely achieved for one particular field, namely the solution of classical field equations for initial-value problems (Roberts and Boris, 1971; Roberts and Peckover, 1971; Kuo-Petravic, Petravic and Roberts, 1971). The method is known as Symbolic Algol and will be described in detail at this Seminar Course by Dr Petravic and Dr Kuo-Petravic.

Fortran, and more particularly, Algol 60, were designed for this dual purpose but have two major weaknesses; firstly they do not include much of the notation that theoretical physicists normally use, and secondly they have no power of extension other than through the use of subroutines or procedures.

We have however been able to show that by writing Algol programs in a particular way, they can be brought into very close correspondence with the notation of vector analysis. For example the magnetic diffusion equation

$$\frac{\partial B}{\partial t} = \mathrm{Curl}(V \times B) - \mathrm{Curl}(\eta\ \mathrm{Curl}\ B) \tag{10}$$

can be programmed in Symbolic Algol I as

```
AB[C1,Q]:= B+DT*(CURL(CROSS(V,B)) - CURL(ETA*CURL(B)));          (11)
```

independently of the coordinate system and of the number of dimensions. Most of the notation is obvious but it should be explained that the prefix 'A' denotes 'array', C1 stands for the current component (or the first component of a tensor), while Q represents the local mesh point at which B is being evaluated.

Modularity has been achieved because the same statement (11) will work just as well for spherical polars as for a Cartesian system, if one simply 'plugs in' or 'switches on' a different definition of the CURL operator. The definition of CURL in Cartesian coordinates is

```
real procedure CURL(A); real A; CURL:= RP(DEL(RP(A)))-RM(DEL(RM(A)));   (12)
```

which gives some idea of the conciseness of Symbolic Algol I as well as of its similarity to the notation of theoretical physics. Here RP,RM are mutually inverse rotation operators, permuting the vector components (123) in the positive (231) and negative (312) directions respectively, while DEL is a vector finite difference operator.

Symbolic Algol I executes quite slowly because of a large number of nested procedure calls. To get around this problem, we have shown that statements such as (11) can be converted either automatically or by hand into an equivalent form called Symbolic Algol II, which when executed will automatically generate an optimized program in any desired output language. For this purpose they are plugged in to an Algol program called the Petravic Generator which is supplied with modules analogous to (12) in order to define the difference scheme, coordinate system, target language and so on which are required for the particular job.

The target code is about as fast as well-written Fortran, and an added advantage is likely to be that code can equally well be produced for computers for which no compiler is yet available, or even for which Fortran is not particularly suitable, such as the new CDC STAR-100 which is able to process complete vectors in one operation without using a DO loop.

What we are doing here is to use the computer itself to write the program, instead of writing it by hand. Since much of the work is tedious and mechanical this is a very natural development, but the interesting point is that the instructions which must be fed to the computer to make it carry out this task are in virtual 1-1 correspondence with the original mathematical statement of the problem. This is a situation which is reminiscent of both quantization and second quantization, in which the equations always seem to remain the same but get interpreted in different ways. If it can be exploited further we may be able to use much of the formalism of mathematical physics itself as the algorithmic language for programming computers.

In this sense I believe that one of our immediate aims should be to weld mathematical and computational physics into a coherent whole.

REFERENCES

Ball W W Rouse (1908); A Short Account of the History of Mathematics, reprinted Dover Publications Ltd., New York (1960).

Barbe P (1970); Software Engineering (Edited J T Tou) 2 Vols. Academic Press, New York & London, Vol.I, p.151.

Bemer R W (1970); Software Engineering, Vol.I, p.121.

Betcher R and Criminale W O (1967); Stability of Parallel Flows, Academic Press, New York and London.

Clementi E (1970); CRPS p.437

David E E Jnr (1969); Software Engineering, Report on a Conference sponsored by the Nato Science Committee, Garmisch, October 1968, p.62.

Elgot G and Robinson A (1964); Journal of the Association for Computing Machinery, 11, 365.

Emmons H W (1970); Critique of Numerical Modelling of Fluid-Mechanics Phenomena.

Fernbach S and Taub A H (1970) (editors) Computers and Their Role in the Physical Sciences*, Gordon and Breach, New York.

Goldstine H H (1970); CRPS p.51.

Goldstine H H and von Neumann (1963), in John von Neumann, Collected Works, Vol.5, Pergamon Press.

Harlow F H (1970); Numerical Methods for Fluid Dynamics, An Annotated Bibliography. Los Alamos Report LA-4281.

Iben I Jnr (1970); CRPS p.595.

Kowarski L (1970); CRPS p.479.

Kuo-Petravic G, Petravic M and Roberts K V (1971); Automatic Optimization of Symbolic Algol Programs, I. General Principles (submitted to Journal of Computational Physics).

McClure R M (1969); Software Engineering, Report on a Conference sponsored by the Nato Science Committee, Garmisch, October 1968, p.66.

Mock O, Olsztyn J, Steel T, Strong J, Teitter A, Wegstein J; (1958), Comm. ACM.

Morse P M and Feshbach H (1953); Methods of Theoretical Physics, 2 Vols. McGraw Hill, New York.

Pasta J R (1970) CRPS p.203.

Poole P C and Waite W M (1970); Software Engineering, 1, 167.

Roberts K V (1969); Computer Physics Communications 1, 1.

-/ ...

* Referred to as CRPS.

Roberts K V and Boris J P (1971); The Solution of Partial Differential Equations using a Symbolic Style of Algol, Journal of Computational Physics (to be published).

Roberts K V and Peckover R S; (1971) Symbolic Programming for Plasma Physicists, Culham Laboratory Preprint CLM-P257.

Snyder J N (1970); CRPS, p.463.

Turing A M (1937); Proc.Lond.Math.Soc. (2) 42 230.

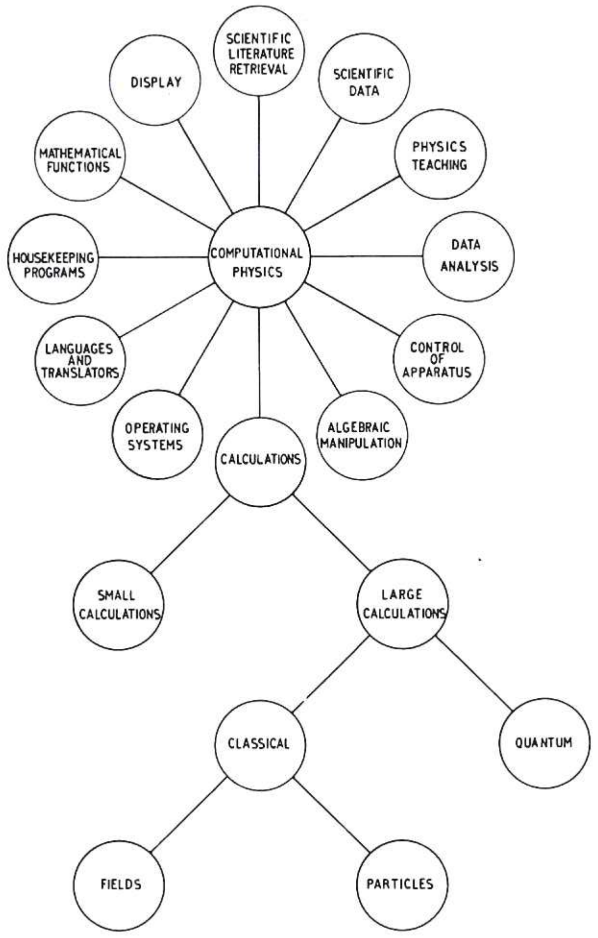

Fig. 1 Some of the areas in which computing has an impact on physics.

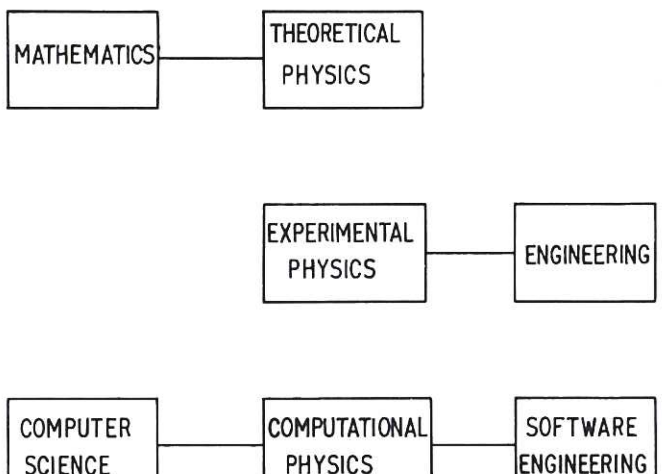

Fig. 2 In the past the progress of mathematics and theoretical physics have been closely associated with one another and similarly for experimental physics and engineering. Computational physics requires advanced techniques in computer science and software engineering and in turn may be expected to contribute to these disciplines.

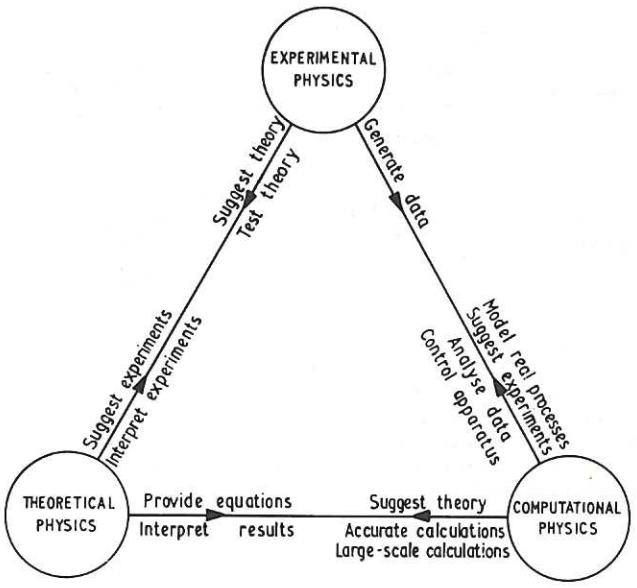

Fig. 3

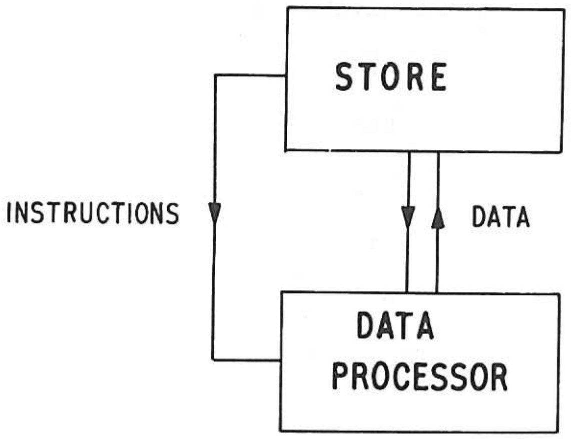

Fig. 4 The computer processes data by means of a sequence of instructions, both being drawn from the same store.

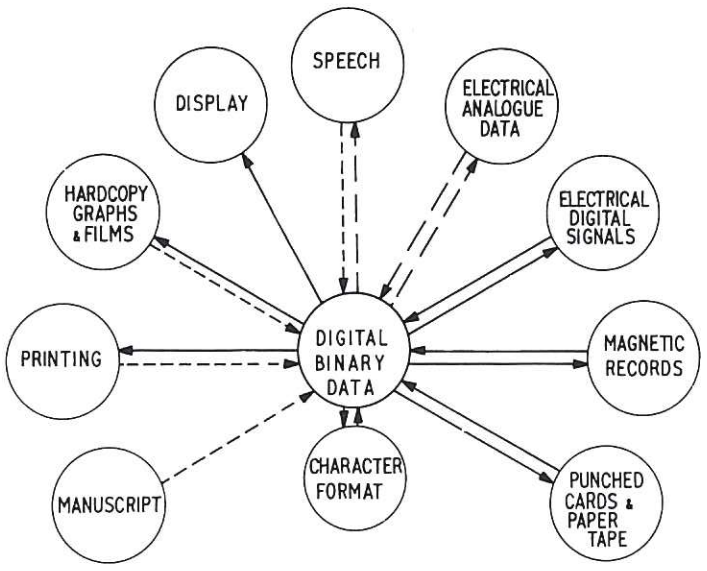

Fig. 5 All forms of digital data are freely convertible into one another (full lines). Printing and display are also straightforward. Analogue-digital conversion can be carried out without difficulty and a computer can be made to talk (long dashes). Those transformations which are represented by short dashes are much more difficult to carry out and should be avoided where possible by storing all data in digital form.

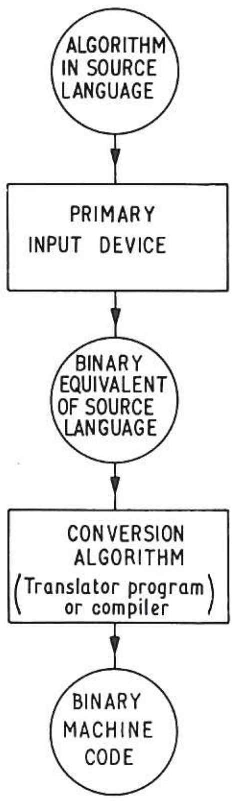
ALGORITHM IN SOURCE LANGUAGE
PRIMARY INPUT DEVICE
BINARY EQUIVALENT OF SOURCE LANGUAGE
CONVERSION ALGORITHM (Translator program or compiler)
BINARY MACHINE CODE

Fig. 6 An algorithm can be expressed in any 'source language', a <u>second</u> algorithm or sequence of algorithms being used to convert this automatically into binary machine code.

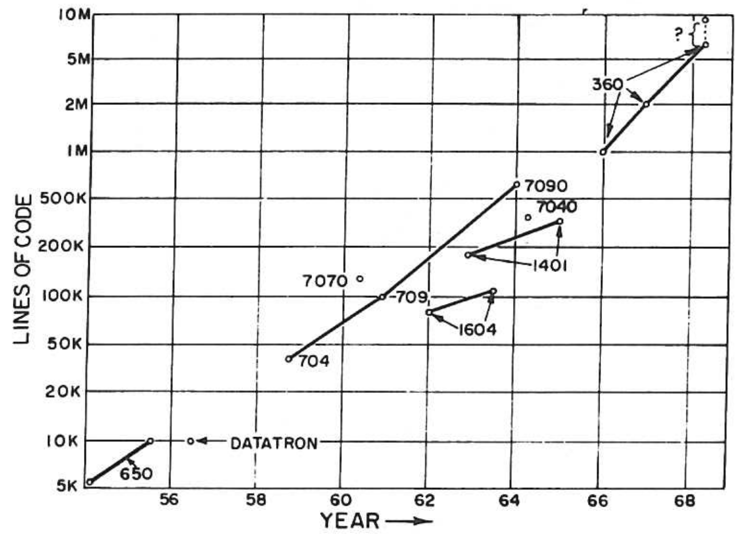

Fig. 7 Exponential growth of the software required by operating systems (McClure 1969). In 10 years the amount of software associated with a typical scientific computing system has increased by a factor 200.

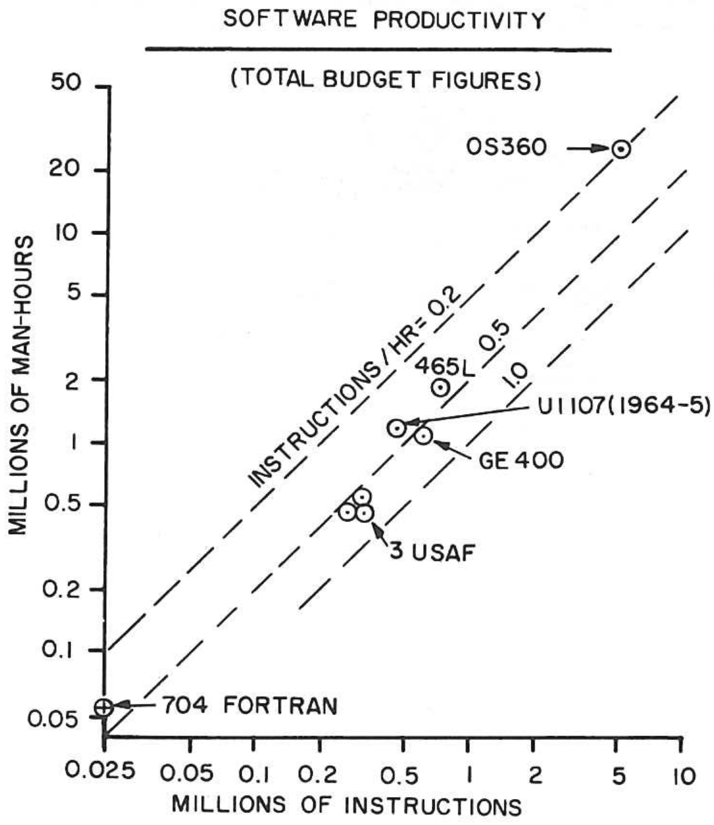

Fig. 8 Effort spent on software construction (Bemer 1970). Because productivity has not increased with time, both the cost and the effort required for software construction are increasing exponentially at a similar rate to that of Fig. 7.

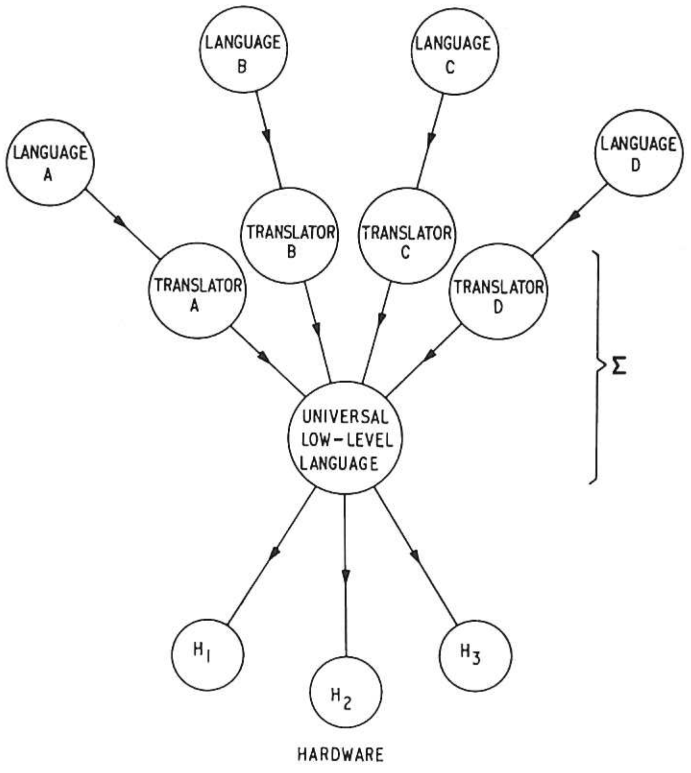

Fig. 9 It is now difficult to introduce new scientific programming languages because of the need to reach agreement on standardization and the cost of constructing a new compiler for each type of computer system . A better solution would be to construct just one compiler which would then be published. Both the compiler itself and its target code must be expressed in a suitable universal language so that they can be used on any system.

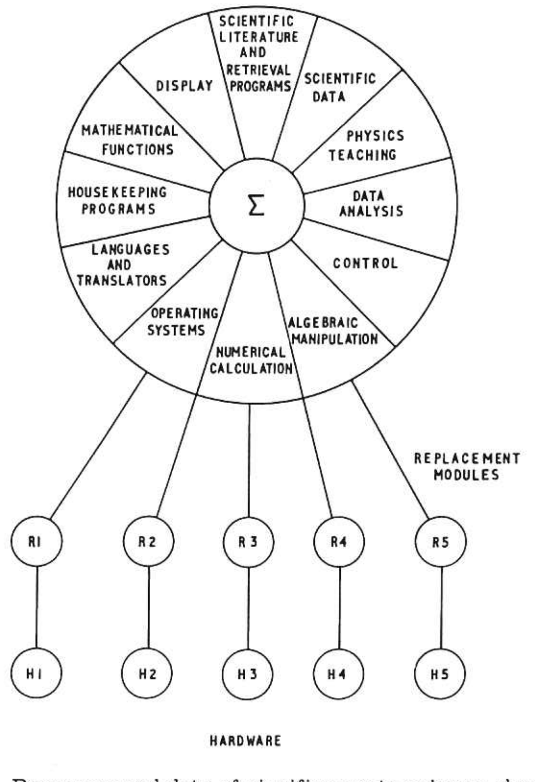

Fig. 10 Programs and data of significance to science should be published in digital form to ensure that they are freely available and that their efficiency and reliability can be checked. This 'digital literature' $\sum$ should include not only scientific applications programs but also operating systems and compilers. Most of it should be expressed in universal form so that it can be used on any machine. Replacement modules R_i written in assembly language are used in the interests of efficiency for those modules which have a high execution frequency.