STANFORD ARTIFICIAL INTELLIGENCE PROJECT
MEMO AIM-146

COMPUTER SCIENCE DEPARTMENT
REPORT NO. CS-224

PARALLEL PROGRAMMING

BY

ANDREI P. ERSHOV

JULY 1971

COMPUTER SCIENCE DEPARTMENT

STANFORD UNIVERSITY

STANFORD ARTIFICIAL INTELLIGENCE PROJECT
MEMO AIM 146

JULY 1971

Parallel Programming

by

Andrei P. Ershov

ABSTRACT: This report is based on lectures given at Stanford University by Dr. Ershov in November, 1970.

Introduction

The author is deeply convinced that programming problems are decisive In the overall design and utilization of parallel computing systems, Of course, the engineer using 'his 'imagination can fairly rapidly propose one or another design for such a computing system and even bring It into being; he does this through analogy or, conversely, through the method of selecting a contrasting design. Thfs approach, however, necessarily leads to a situation In which programming problems become secondary, and the degree of freedom available in solving the mis automatically limited by a prescribed structure that is frequently not responsive to essential points.

'The problem of programming is of essentially secondary importance when one speaks of a particular task that must be solved on a given machine, However, the problem of programming is of primary Importance in any thoughtful approach to the design of a computing system, in the sense that the developer must by the strength of his own penetration into the essence of the matter mentally include in the design the complete set of all possible programs and algorithms, On the basis of an analysis of their internal properties, structures, and external characteristics, he formulates design principles tot the computing system, These principles must be adequate for the examined properties of the programs, and--most Important-must be appropriate for reducing the programming task to a formal procedure and for automating It.

Automation of programming for Parallel computing systems Is a basic necessity for a number of reasons,

First, the description of a parallel program is not characteristic of the nature of man's algorithmic thought processes. In fact, this Is true only to the extent that we are considering the implementation on a computing system of a task specified in algorithmic form, Today this Is the most common way of stating a problem; however, in the future this could change, since parallelism in a problem is frequently implicit in its initial formulating and is artificially driven out by algorithmization, This will be discussed In more detail below,

Second, Independent of whether man thinks "sequentially" or "in parallel," he tends to think laconically-to seek compactness In problem formulation, At the same time, studies on the design of parallel programs indicate that in many cases complete parallelism can be achieved only under conditions of potentially unlimited "multiplication" of some initial program constructions, where the multiplication is associated with systematic modification of the constructions, Thus, some portion of the programming process becomes an organic part of the oomputational process and therefore must necessarily be transferred to the machine,

In this work an attempt will be made to demonstrate the pre-eminence of programming over hardware design. of a computing system, The concepts of parallel programming established at the present time will be examined, and on this basis an evaluation will be made of several approaches to the structure of computing systems. Due to the general laok of development of the problem, this analysis will be superficial and in places even speculative, However, the fog that obscures our ultimate goal can be dissipated only if we boldly step out Into It, On the other hand, an attempt will be made to put forth some recommendations from a realistic position, taking Into consideration contemporary technical capabilities, historically established frameworks, and other current limitations. Finally, the paper is limited in the sense that it is not a balanced, inclusive survey, but only reflects the personal viewpoint of the author, nurtured however over a long period of time,

Basic Concepts Of Parallel Programming

In any theory of parallel programming aspiring to completeness, there are three components that must be present:

--a way of describing a parallel program (parallet programming facilities);
--a way of deriving a parallel program from ordinary algorithmic notation (desequencing of sequential algorithms);
--a way of executing a parallel program (specification of the computing system, assignment of processors to jobs, control of the parallel computational process),

The recommended structure for a computing system must be derived from basic principles reflecting the most essential aspects of each of these components,

Three approaches to parallel programming can be examined, and will be conditionally designated as follows:

- natural Parallelism,
- concurrent computations in algorithmic languages,
- asynchronous programming-,

Natural parallelism is far from what might be somehow considered a comprehensive theory. However, some of its fundamental considerations are quite clear and in many respects highly attractive and distinctive, giving Us the right to regard It as 4 basic approach, This theory does not have a specific advocate, although it is carried *on* the wind and appears tacitly In many works, Therefore, it is difficult to clearly specify sources for it,

The basic thesis of this theory is that in nature everything in fact takes place In Parallel, and the more literally some actual process is described, the closer such a description is to a parallel program, The general problem is that one must be able to successfully and simply capture this natural parallelism and describe

it In some universal System of concepts, based on such principles a3 simultaneity of many events, proximity of actions, etc.--i.e., a system of concepts in which a phenomenon is represented as a Set of many elementary processes occurring in a space simultaneously and Interacting oniy with neighboring processes. The principle of determinism (Indeed understood in a broad sense) Is the basis of this approach; It say3 in' the given case that if a phenomenon at some moment Is described as a set of initial states and mechanism3 Of Interaction, then the entire subsequent history is predetermined and can be observed, measured, and computed. The space referred to does not necessarily have to be physical and continuous-it can be discrete, finite, etc.; it is only important that the concept of the Immediate neighborhood be determined.

This principle is quite well known and In fact we apply it In many situations (in particular, ail methods of describing phenomena in differential form are based on It), The real and far from trivial problem is to constructively formulate this principle in a universal form representing all the essentials that must be embodied in parallel programs, in view of the diverse computational universes in which Informational models of phenomena of interest to us may be embedded,

On the whole, programming under such an approach disappears as a separate task and becomes an Inseparable constituent and maybe even the whole of the Problem of describing the Initial situation for the given phenomenon.

Under a pure formulation of this question, the problem of translating "ordinary" notation into a Parallel one also does not arise, since natural parallelism is primary and the concept of "ordinary" notation loses Its meaning, However, a pure formulation of the question is not possible, since the entire computational aspect concerned with the design of the arithmetic model of the phenomenon bears imprints of the inherently sequential processes of logical reasoning which are the foundation of any algorithm, Therefore, any systematic attempt to construct a direct bridge from natural Parallelism to Its Impiementatfon In a computing system requires an essential revision of all numerical analysis and the apparatus of mathematical physics. An additional and no less vast problem Is the design of such a system of concepts in which any Interaction occurring In nature is described as a process of Information exchange, and in which the Interacting element of a space Is represented as a miniature, universal computer with many Input and output lines doing the required information Processing,

If we turn to the third component of natural parallelism, then It Is Immediately necessary to make a hypothesis about an appropriate computing system, It is perfectly obvious that a computer environment* [1] is the first candidate for such a system, A consideration of such an environment properly is deferred until the conclusion of this paper, but It should be mentioned here that if there is enough computing power, we can associate a separate unit of the environment with each element of the space, all control and

implementation of the computational process reduces to the construction and start-up of the environment, The environment reproduces the described phenomenon almost "one-for-one", and thereby frees us of concerns about organiring the computational process,

However, the Problem becomes substantially more complex if the parallel process consists Of Just one element more than the available number of units of the environment, Very complex problems of accounting for boundary effects and of rearranging the environment during implementation of the computational Process then arise,

Concluding this consideration Of natural parallelism, we note that we began with a statement as to the attractiveness of the approach and then turned attention to the fact that with it some problems disappear, However, the entire remaining discussion was directed towards discrediting the approach, The discreditation lies in the fact that for development of the computational environment as a universal approach it is to o revolutionary, breaking with many past concepts and leading to many problems on the level of fantasy; therefore, It cannot be recommended as a guide for immediate action,

At the same time, the author hopes that this discreditation does not result in any weakening Of the search for a general-purpose scheme for introducing natural parallelism Into programming, In particular, this must not occur because successful synthesis of natural parallelism and a computational envlronment in efficiently operating technical devices would have a decisive result on research into the structure and operating mechanisms of the brain--the most perfect computational environment created by nature,

Concurrent computations In algorithmic languages, in comparison to the approach Just examined, lower us from the skies to the earth, This approach represents a somewhat minimal attempt to introduce parallelism Into algorithmic languages, fully taking into account existing technology, contemporary machines architecture, and the organization of Operating systems, The approach puts aside many questions O f parallel programming, but in return it permits us to use, even if only partially, those capabilities that today's techrology represents, .

The descriptive facilities for denoting parallelism, about which we speak in this section, are found in such languages as those for simulation (SIMSCRIPT[2], SIMULA[3]) and the "new generation" of algorithmic languages--PL/1[4], ALGOL-68[5], The existence of these parallel, or concur rent, computational branches are clearly explicated In an algorithm, Branches parallel to one another usually have a common beginning and end, Each branch, in turn, can consist of sub-branches, Thus, ail program parallelism Is explicitly stated, and the points of branching and joining can be determined

*Translator's note: "environment" Is used as a translation of the Russian word "sreda" which is evidently used as a technical term In the paper referred to,

syntactically. Branches can use common variables and can include means for interruption or suspension of branches depending on the value of the variable quantity. Thus, synchronization of concurrent computations is accomplished, the explicit programming of which is a task for programmers.

The described approach still does not offer in its general notation any way of dividing the program into parallel branches, since use of the descriptive facilities of parallelness is not formalized and is completely up to the programmer.

In the majority of cases, the introduction of parallel branches permits parallel execution but does not prescribe it. In the case of a deviation from full parallelism, a well-written program permits the branches to be executed in arbitrary order, which eases the work of the operating system. However, in truth the existing languages do not give formal rules for defining a "well-formed" program.

The organization of computations for a program with Parallel branches takes place according to the Principles of multiprogramming, where the program for solving a large problem is represented to the "eyes" of the operating system supervisor as a stream of related small Jobs. One parallel branch is represented as One Job. A kind of passport is set up for this Job, and stored in the operating system. When the control reaches the branching point, a Call (established there by the translator) to the supervisor is executed, which activates the passports of the branches beginning at that point and makes them candidates for processing. The supervisor carries out some planning at the assignment of branches for processing. Its purpose is to maximally load units (or processors) that are able to operate together, and it attempts to achieve completion of all parallel branches as rapidly as possible, since only a concurrent output at the convergence point will permit an advance to the next point in the program.

This method is practical only when points of branching and convergence occur relatively infrequently. This at once imposes a serious limitation on the degree of parallelism and on the number of permissible branches. Another limitation is that the parallel structure of the program is static and in fact does not permit, for example, expansion of loops.

Asynchronous programming, from the author's viewpoint, has the most right to be called a theory of Parallel programming, and therefore will be considered more thoroughly. The first results of the six-year work of the young mathematicians V.E. Kotov and A.S. Narin'yani form the fundamentals of this theory [6-9]. The work of the American mathematicians Karp and Miller [10] is related to this direction. Asynchronous programming from the very beginning has developed as a mathematical theory, based on certain axiomatics. And in this lies its merit and its imperfections. No computing systems have yet been designed or translators created according to the principles of asynchronous programming. But the theory of

asynchronous programming promises to:

--provide a precise concept of a parallel (asynchronous) program;

--describe a class of computing systems;

--rigorously define program equivalence;

--formulate criteria of the correctness of parallel programs for a given system;

--define degrees of parallelism and maximum parallelism;

--constructively prove the possibility of formal translation of an ordinary program into a parallel program possessing maximum parallelism;

--study certain internal problems of parallel programming.

An additional characteristic of asynohronous programming is that in its theoretical aspect it is linked with the already rather well-developed theory of sequential Programs--Program schemes [11], this association is still not fully developed, but the prospects of doing so are clear,

we shall examine the fundamental concept9 of asynchronous programming, The initial concept is that of a quasiprogram, which is represented as an arbitrary set of statements operating on a memory which consists of variables, A statement is a multi-pole "black box" which assigns output values to its outbut line9 as functions Of its inputs, Associated with each statement input and output pole are variables, The input variables supply the statement with values from memory, and the output variables accept the outputs of the statement, Each statement can be In one of three states: dormant, ready, or functioning,

A quasiprogram is executed In a computing system In the following manner, At the moment of start-up, the initial state of memory is assigned, All statements are dormant, The operation of the system consists of a succession Of changes In the states of the computation at separate, discrete moment9 of time, At each such moment, the computing system can transfer Statements from one state to another, with only a ready statement able to become a functioning one,

If a statement becomes functioning, it receives at that moment from memory values Of arguments, If a statement stops functioning, It transmits to memory at that moment the values of computed results,

The progress of the computations can be depicted graphically In the form of a computational process, represented as a time diagram of the switching Statements on and *off*, whereby "on" is understood transfer to the functioning state, and by "off" is understood exit from the functioning state, An interval of the time diagram spanning moment9 of on and Off switchings reflects one statement action (Fig. 1),

From this computational process it is possible to

unambiguously plot its Important characteristics--to construct an information-logic graph, i.e., this oriented graph is without cycles, and its vertices are the statement actions (Fig. 2). From vertex A there is an arc to vertex B if at action B some of its arguments receive information assigned by some result of action A, or if action A in a direct way has an effect on action B. The concept of a direct effect of one statement on another is quite complexly determined, but the example of Fig. 3 gives some impression of its nature. Here, statement S_1 directly affects statement S_2, whereas S_1 has no effect on S_3. Statement S_0 affects S_2 only indirectly, and S_1 and S_3 directly.

The information-logic graph, on the one hand, retains some necessary minimum of information that makes it possible to re-establish the way of processing the input data of the computational process into the final results. On the other hand, it ignores less essential details of the progress of the computational process.

The fundamental characteristic of asynchronous programming is the assumption of non-uniqueness of the execution of the computing system to a quasiprogram. Generally speaking, it is assumed that the system has some parameters or degrees of freedom that are not fixed by the quasiprogram. This means that for a given quasiprogram for some initial memory state, the system can implement some set, possibly infinite, of computational processes.

A quasiprogram is called a program for a given computing system if for any assigned initial memory state all computational processes implemented for it by the system have the same information-logic graph. Thus, an information-logic graph is invariant, guaranteeing accuracy of information processing under any behavior of the computing system executing the given program.

The computing system's greater degree of freedom is due to the following. It is assumed that each statement in a program is equipped with a predicate that has some inputs from memory; this predicate is called a trigger function. The statement and its trigger function are called a block. In executing the program, the system continuously computes the values of all trigger functions of the program blocks. Blocks with trigger functions equal to 1 are considered to be ready, while the others are dormant. At any moment in time, the system can switch any ready blocks on or switch any functioning ones off. Computing systems of this type are called asynchronous systems.

Thus, we see that asynchronous programs have a very important quality: protection from arbitrariness that the system may exhibit with respect to the moments of switching of ready operators on, the time of their execution, or the amount of computing facilities available for appointment of the ready operators for operation. An asynchronous program does not impose any specific requirements for the computing system regarding its time characteristics or computing power (number of processors). Moreover, system characteristics can

be altered dynamically, without any loss of program runvalidity.

The concepts of the degree of parallelness, or the degree of asynchronousness, are introduced in asynchronous programming in an interesting manner, Quantitative measures of asynchronousness are not used; however, it is possible to determine that one program is more asynchronous than another, and also to determine the most asynchronous program among a set of programs being compared for asynchronousness,

We shall consider two programs, P↓1 and P↓2, equivalent in the sense that for identical initial memory states they generate computational processes with identical information-logic graphs, For each information- logic graph IL there are, correspondingly, sets M↓1 and M↓2 of the computational processes possessed by such graph IL, Thus, if for any IL, M↓1 ⊃ M↓2, then P↓1 possesses greater asynchronousness, In this way, the Program that allows the computing system greater flexibility jn the range of computatjonaj processes resulting In the given Information-graph Is recognized as being more asynchronous, There are no explicit statements about more or less parallelism, and this is correct since the degree of parallelness Of a program is actually determined not only inherently, but also by the capabilities Of the system, However, If program P↓1 can be executed by all the same ways as can P↓2, and In addition by still other ways, then there is some chance that It has greater parallelness, and anyway not less,

The degree of asynchronousness also has a more constructive form of determination, The degree of diversity in executing programs in the final analysis rests on the set of binary relations among program statements for determining whether Or not restrictions exist on the relative order of statement execution, the sets of these binary relations forms a dependence graph, The fewer edges the dependence graph has, the more asynchronous js the program, If the Program has a dependence graph such that removal of any edge from jt clearly destroys the information-logic connections of some execution of the program, then such a program has maximum asynchronousness,

AS already mentioned, the theory of asynchronous programming offers a constructive method for formal transfer from sequential programs, considered in the form of program schemata, to asynchronous programs of maximum parallelness, This transfer takes place In two stages, In the first stage, several equivalent transformations of the scheme jtsejf occur, and then a single transformation of the scheme into an asynchronous program Is performed; the latter consists of "splitting" the scheme Into the Individual statements and of assigning for each statement its trigger function,

I t appears that a barrier in the path Of converting from a program scheme to a maximally asynchronous program lies In the fact that the Internal asynchronousness of a Scheme can be proven to be actually less than its potential asynchronousness, This difference results in an unsuccessful allocation of memory, Imposing artificial connections between statements, and In Imperfect multiplication O f

some Statements into several Instances which, when executed simultaneously, can extend the rang8 of Program executions. Two Pairs of statements, depicted in Fig, 4, serve as an example Of the difference between internal and potential asynohronousness, Since the same value y is used in each pair to transfer information, these two pairs of Operators cannot be executed concurrently, which would be permissible if jn each pair its own variables were Used,

In the work of V.E. Kotov, a methodology is offered for transforming any Program scheme Into an equivalent scheme, the internal asynchronousness o f which reaches its potential asynchronousness,

In the development of a theory O f parallel programming, I t was unclear earl ier whether or not, without sacrificing generality of results, quantitative Indicators of the passage of time could be included, threading o n the temporal axis only the fact Of the occurrence of events, and not their duration, The assumotfon that all changes In state occur instantaneously, Including the actions of statements, Is the extreme expression of this abstraction, It has been shown by A.S. Narin'yani that for any Information-logic graph there exjsts an executable computational process In which all statements activate instantaneously (the so-called reduced process), Moreover, it has been shown by hjm that if a cauasiprogram Is a Program for a computing system with Instantaneous action O f statements, then I t will also be the program for a oomputing system with any prolonged action of statements,

Contjnuous computation of trigger functions occupies a significant place jn the execution of asynchronous programs, If such a process Is to be implemented In a real system, maximum simplification and accejeratjon of this computation becomes very Important, From this viewpoint, the result obtained by Z. Zvinogrodskij* is very Interesting, It shows that for any asynchronous program it i s possible to constructivejy find its equivalent program such that all trigger functions In that program have the form of logical functions o f elementary conditions expressing only the fact of the action of the operator, This makes It possible Without loss o f generality t o attribute to the computation of trigger funotjons a specific appearance that permits realization using interrupt registers and other high-speed hardware facilities,

* Private communication

Discussion o f the Structure O f Computing Systems

the US8 Of binary switching and the realization *of* Boolean functions remain for the foreseeable future the f jrm foundation O f electronics, based In its most direct form on the principal of discrete actions, However, the "embedding" Of elementary binary structures within larger structures reflecting a n a priori

algorithmic basic of programming must be one of the fundamental principles for achieving pre-eminence of programming over hardware. The author has already mentioned that the established algorithmic basis (arithmetic actions and relations on numbers, associative and addressed information search) has an historical nature and, possibly, is subject to revision from the tenets of natural parallelism. Nevertheless, we do not presently see the possibility of stepping beyond the limits of this algorithmic base, which, to a point, is taken fully into account in contemporary algorithmic languages and the axiomatics of the theory of programming,

Therefore, the author specifically does not recommend homogeneous binary computational environments with . dynamic restructuring as the primary candidate for a computational universe, Because of its universality, any such complex will lose out in productivity to a computing system in which the structure of the algorithmic base (accumulators, multipliers, index registers, address matrices, etc.) is introduced a priori at the moment of design and is implemented with the assistance of a rich arsenal of special devices,

In particular, the author considers advisable the preservation of the established separation between the organization of information processing In active Processors and its Storage in a passive store, A large volume of a common memory with random access is the most suitable unit for programming, "absorbing" and decoupling all difficulties concerned with boundedness, with the necessity for fast commutation of processors far some operation In a sequence that was not predictable earlier,

Distribution of memory among processors leads to the necessity of establishing more rigid synchronization, In fact, If processor A transfers results of its operation to another processor B, then this means that the results at A consist of only part of that fnformation which is needed for the operation Of B (if this were not so, then A could continue the operation itself, without transferring It to B), But consistent receipt of information by processor B from various parts of the system can lead to a conflict with the desire to utilize as fully as possible the productivity of the processor and to not create downtimes, since generally speaking It is possible to guarantee such productivity only under "equi-accessibility" Of any operation to any processor, which contradicts the principle of memory distribution, Anothar threat to productivity from memory distribution is that in order to provide remote data transmission connections, too many processors must operate only as transit points, Thus, a homogeneous system of many processors with a distributed memory becomes economfcaiiy justified, in the opinion of the author, only when the utilization factor for the processors may be comparatively low and is measured by roughly the same values as the utilization factor for the core memory locations, determined by the number of location accesses relative to the duration of the storage of information in the location,

Thus, as a primary candidats for the computing system of the Immediate future, the author advocates a homogeneous computing system

on a common memory wfth sufficiently well developed processors structured on an established algorithmic base. This reasoning, however, is in need of some close examination.

First, the programming for such a system must be based on the principles of asynchronousness, since the architecture of this type of system is most adequate to an abstract model of the computing systems used in asynchronous programming theory. In addition, program asynchronousness makes very simple the problem of assigning processors to operations, including dynamic assignment, if only the problem of fast scanning of the program's trigger functions is solved.

Second, the system must include the principle of separating strictly computational processes (execution of program statements) from control processes (scanning trigger functions or external interrupt signals and assigning processors to an operation).

It is very tempting to concentrate the control process in a special processor, called here the monitor. Processor general registers must be accessible, on the one hand, to the monitor, while on the other hand the monitor must have in its own memory the complete "logical scheme" of any program presented to it in its capacity as a generator of a flow of' requests for a Job to be executed. Processing these requests by the method of assigning processors to various jobs constitutes the essential aspect of control.

Such an approach is of interest also because, it seems to the author, it permits the union within the framework of a single system architecture of what would appear to be contrasting types of operations, such as the organization of multiaccess use (i.e., allocation of the computational resources among a flow of weakly connected service requests), and parallel, multiprocessor operation (i.e., concentration of the computational resources for the solution of one large problem). This unity is achieved by the fact that if we examine the "anatomical structure" of the process of controlling the solution of a large problem written in the form of an asynchronous program, then this structure-would appear as a limit case of the flow of requests for operations in a time-sharing system.

Separating control out as an individual process, serviced by the monitor, has its own weaknesses. One of these is that critical requirements are imposed on the speed of scanning trigger functions, assigning processors to an operation, and loading processor register stores. Another factor is that centralization of control reduces system reliability. A different scenario is one in which each processor itself searches out its own new job as soon as it finishes its existing operations. This requires the creation in the common memory of a special "labor exchange", access to which is open to every processor. This exchange can be duplicated or can have "branches" in segments of the store, preferably those associated with given processors.

Acknowledging that it is premature to make final Judgments on the USE Of centralized or distributed control, the author nevertheless considers that it is necessary to fully carry out the development of methods for centralized control, A consideration Of control as a self-contained function, suitable also for special hardware implementation, has particular significance in any objective solution of the problem of the relationship between programming and hardware facilities for exercising control over a computing system, Operational experience with well-developed operating systems for both batch processing and multiaccess use has indicated that the control processes have their own algorithmic base, the structure of which can and must be embedded in special hardware, This does not exclude, however, the fact that this currently specific base will eventually merge with the "general algorithmic" base, However, this should not be taken for granted; it would have to come about as the result Of solving a clearly Stated problem,

Conclusian

What, from the viewpoint of the author, must be the fundamental directions for work in the area of programming for parallel computing systems?

First, it is necessary to implement in contemporary algorithmic languages such parallel programming facilities as parallel branches of concurrent computations,

On the basis of the methodology of asynchronous programming, It seems that one could supplement "manual" methods by facilities for automatic determination of parallel branches for reduction of the number of parallel branches, making remaining ones longer, and the checking of a Parallel program's asynchronousness, These techniques, by the mid-seventies, could be fully implemented on even operating systems for third-generation multiprocessor computer configurations, Some fundamentals for such techniques can be found already in our existing operating systems [12],

The second direction, which the author believes can provide the richest results by the end of the seventies, must be for a comprehensive design of asynchronous programming as a completed theory, linked with the formal theory of sequential programs in that it would become a working instrument of Parallel programming,

Efforts must be concentrated on solution of the following Problems:

--the mechanism for computing trigger functions;
--derivation of the "logical scheme" of a parallel program during the translation process and its dynamic augmentation or alteration during program running;
--algorithms for dynamic and static assignment of processors to an operation;
--methodologies for fast commutation of processors during the

expansion of sequential cyclic computations;

--organization of buffering and dynamic loading of a faster store from a slower one;

-study Of the capabilities of distributed and centralized controls;

--finding a special algorithmic base for control processors in computing systems.

Successful progress in these directions, in the author's opinion, will help us by the end of the seventies obtain fourth-generation computing systems with productivity ratings greater than 100-million operations per second,

As the third direction, it is necessary to support research on a broad front into computational environments with fundamental study of the following questions:

--the search for the most appropriate universal computational cell and determination of the degree of its connectivity with the environment;

--study of boundary effects in bounded environments and the Problem of dynamic restructuring of the environment;

--research fnto the capabilities and feasibility of building "multiple", mutually penetrating environments with various functional purposes (for example, control and operating, computing and transporting environments;

--development of concrete special-application devices for problems whose structure is in itself sufficient for the selected environment structure;

--development of the principles of natural parallelism and then on that basis a reconsideration of our algorithmic base,

In the opinion of the author, in the eighties computational environments will become a competitor for conquering the problem of increasing the ceiling on the Power Of computational facilities and the creation of an artificial intelligence,

Acknowledgement

The author deeply thanks Mr, Wad8 B, Holland of RAND Corporation, who kindly agreed to translate this text into English,

References

1. E.V. Evreinov, "On the Microstructure of the Elementary Machines of a Computing System," Computing Systems [Vychislitel'nye sistemy], No. 4, Institute of Mathematics, Siberian Department, USSR Academy of Sciences, Novosibirsk, 1962 [in Russian].

2. H. Markowitz, et al., SIMSCRIPT--An Algorithmic Language for Simulation [SIMSKRIPT--algoritmicheskij yazyk dlya modelirovaniya], Soviet Radio Publishing House, Moscow, 1966 [in Russian translation].

3. O.I. Dahl and K. Nygaard, "SIMULA--A Language for the Programming and Description of Systems with Discrete Events," Algorithms and Algorithmic Languages [Algoritmy i algoritmicheskie yazyki], No. 2, Comouting Center, USSR Academy of Sciences, Moscow, 1967 [in Russian translation].

4. PL/1 General-Purpose Programming Language [Universal'nyj yazyk programmirovaniya PL/1], Mir Publishing House, Moscow, 1968 [in Russian translation].

5. A. Van Wijngaarden, et al., "Report on the ALGOL-68 Algorithmic Language," Cybernetics [Kibernetika], No. 6, 1969; No. 1, 1970.

6. V.E. Kotov and A.S. Narin'yani, "Asynchronous Computing Processes on a Memory", Cybernetics [Kibernetika], No. 3, 1966 [in Russian].

7. V.E. Kotov and A.S. Narin'yani, "On Transtormation of Sequential Programs into Asynchronous Parallel Programs", Information Processing 68, North-Holland, Amsterdam, 1969.

8. A.S. Narin'yani, "Asynchronous Computing Processes on a Common Memory", Candidate Dissertation, Computing Center, Siberian Department, USSR Academy of Sciences, Novosibirsk, 1970 [in Russian].

9. V.E. Kotov, "Transformation of Operator Schemes Into Asynchronous Programs", Candidate Dissertation, Computing Center, Siberian Department, USSR Academy of Sciences, Novosibirsk, 1970 [in Russian].

10. R.M. Karp and R.X. Miller, "Parallel Program Schemata", J. Comp. and Syst. Sci., Vol. 3, 1969, pp. 147-195.

11. A.P. Ershov and A.A. Lyaounov, "Formalization of the Concept of a Program", Cybernetics [Kibernetika], No. 5, 3.967 [in Russian].

12. I.B. Zadykhajlo, et al., "Operating System of the Institute of Applied Mathematics of the USSR Academy of Sciences for the BESM-6 (OS IPM)," Transactions of the Second All-Union Conference on Programming [Trudy 2-J Vsesoyuznoj konferentsii po programmirovaniyu], Session C, Novosibirsk, February 3-6, 1970 [in Russian].