Bull. Nov. Comp. Center, Comp. Science, 6 (1997), 71–85
© 1997 NCC Publisher

WinALT – a simulation system for computations with spatial parallelism

S.V. Piskunov

A description of user interface and language of a simulation system (WinALT) is given in the article. This system is based upon formal model called Parallel Substitution Algorithm (PSA). The system combines the best features of its ancestors. It has extensible functionality and is open for user modifications and extensions. It has a wide set of tools for visualization and representation of massively parallel distributed in space computations.

1. Introduction

Nowadays the most important ways to increase the performance of hardware are based upon parallel computations. One of parallelism types which are practically used is called fine-grained. Fine-grained parallelism is based upon the search of source problem transformations which would convert its solution into a group of simultaneously massively executed processes distributed in discrete (cellular) data space.

The possibility to reach the highest degree of parallelism which can be achieved for a particular problem ("natural" parallelism) has made this type of parallelism rather attractive for different applications. This type of parallelism gives good results when used in almost any multiprocessor system in the event that a source fine-grained algorithm is "built correctly". A correctly built algorithm has the following features. Such an algorithm has a multitude of rather simple operators. These operators perform local and parallel data processing. Cellular algorithms, systolic and cellular-neural algorithms satisfy the limitations mentioned above. Algorithms of these types are used in many practical applications such as image processing, vector-matrix computations, discrete graph optimization, image recognition and so on.

The development and research of fine-grained algorithms are always based upon a certain computation model. Several models of fine-grained computation representation were developed by now. The best-known are the Neumann classic cellular automaton, associative machine, systolic structure and neural network. These models are distinguished by data structures, structure of operators and rules of their application to data. One of the fine-grained parallelism advantages is that many implemented and hypothetic special purpose high performance processors are being a direct hardware representation of their respective fine-grained models.

An effort of a unified formal model creation has resulted into emergence of the so called Parallel Substitution Algorithm (PSA) [1, 2]. PSA incapsulates all previously existed fine-grained models. The creation of this unified model can be justified by complexity which is a habitual feature of almost any practically useful fine-grained algorithm. The fine-grained algorithms can hardly be tackled without computer aided tools for a researcher or a developer. The unified fine-grained model gives an opportunity to develop a single widely used tool instead of many for each particular model. The tools based on this unified theory have been developed for several years [2–6].

The *aim* of this article is to give a brief description of the new simulation system of massively parallel distributed computations (WinALT).

A brief description of WinALT ancestors, which are based upon PSA, is given in this article. An attempt has been made to draw their reciprocal comparisons and to outline their advantages and faults (Section 2). The requirements to the new system were formulated on the base of the previously done analysis. These requirements meet the demands of researchers of big dynamic discrete systems. A concise overview of the WinALT GUI (Graphic User Interface) and language main distinctive features is given (Section 3).

2. Languages and systems of parallel microprogramming

Languages and the tools, presented in the given section, were primarily oriented to the description of the fine-grained algorithms and structures at such level of detailed elaboration, which is referred to as a microprogram level. But, as we deal with parallel algorithms and structures, it is reasonable to name this level as a level of parallel microprogramming [1].

2.1. Brief description of PSA

PSA combines features of cellular automaton and features of the Markov algorithm. In this model the data are represented as array of named cells (cellular arrays). A data item is stored in a separate cell. The data transformation is carried out by a set of commands (parallel substitutions). PSA is characterized by total and simultaneous application of commands to cellular arrays and explicit indication of the spatial relations between cells in commands.

It should be noted, that PSA includes rather widely known model of symbolic substitutions [7], which has appeared later. This is supported by the description of the adder of many positive binary integers [8]. The PSA

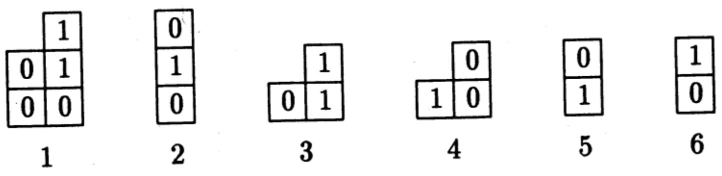


Figure 1

of this adder contains two commands which are represented by symbolic substitutions. These commands perform the transformation of binary digits in the rectangular binary table. The rows of this table are integers for the summation. The binary patterns used in commands are shown in Figure 1. The patterns *1*, *4* form, accordingly, left-hand and right-hand parts of the first substitution, the patterns *2*, *6* represent the second substitution. In the table at each step simultaneously in all configurations of the form *1* and *2* the subpattern *3* in *1* is replaced by the pattern *4*, and the subpattern *5* in *2* – by the pattern *6* and etc., while such transformations are possible. When such transformations are impossible, it means that the sum is calculated.

Of course, the set of means used in PSA is much wider, than it is shown above. In particular, in record of left and right parts of substitutions, along with symbols from some alphabet (we shall designate A), variables can be applied, or some functions with ranges of definition and ranges of values from the alphabet A. These means allow to describe rather complex functional transformation in one parallel substitution.

2.2. Parallel microprogramming language

The first fine-grained (cellular) algorithms and structures simulation language was offered in [1] and was named Parallel Microprogramming Language (PML). It has served as the prototype for all other simulation languages, based on PSA. Conceptually, the language directly follows PSA. The differences from PSA are caused by necessity of the exact description of syntax, as the language is intended to be used on a computer. It is also taken into account, that the parallel work of algorithms and structures is simulated on a sequential computer. An alphabet of cell states, a cellular array, a template, a functional transformation executed by a cell, a substitution microcommand, a cyclic block, a tool microprogram, a parallel microprogram serve as the basic constructive elements of PML.

The alphabet of the cell states consists of two groups of symbols: basic and variable. The basic symbol is represented by either figure, or identifier, consisting of the letter and probably empty set of figures. The basic symbols can also be represented as codes. These codes are symbol strings in a certain base alphabet, usually {0, 1}. The set of basic symbols can be dissected on classes. The names of classes form group of variable symbols. A variable symbol is represented as a basic symbol but unlike the latter it has only a symbol of prefix $. It is also possible to use the codes for representation of variable symbols.

Each cellular array has a name. The names of cellular arrays are unified. Each name begins with prefix TS, which is followed by a positive integer. The array dimension cannot exceed three. The array dimension is specified by the list of the coordinate axes names (indexes) `I`, `J`, `K` in brackets after array name. The size of each cellular array is listed in a microprogram head part, which contains the declarations. A state of an array cell is a basic symbol or its symbol code.

It is evident from Subsection 2.1 that essence of PSA consists of parallel search in a cellular array of copies of some sample and subsequent initializing the found cells with new states, taken from another sample. Both parallel search and the following states assignment can be performed at once for the whole set of various samples. The samples are taken from the left-hand and right-hand parts of substitutions. Syntactic constructions, which reflect this essence, are realized in the language. A construction, which describes a sample, is named a zone. The microcommands are built up from the zones. The cyclic blocks are built up from the microcommands. The parallel microprograms are built up from the cyclic blocks and separate microcommands.

The use of a zone, specifying a sample with the help of a shift pattern, is most typical in the language. The shift pattern is a list of elements, each of which contains a number of integers that is equal to dimension of a cellular array, for which the pattern is intended. Always an element of the pattern exists, which contains only zeroes. The elements of the pattern are initialized with numbers. A pattern name begins with prefix `PAT`, which is followed by a positive integer. The pattern serves for selection of groups of cells in a cellular array. The procedure of coordinates calculation of group cells in a certain (in a general case) 3D cellular array consists of summation of the same triplet of values of indexes `I`, `J`, `K` with constants from pattern elements. One can see that a set of shift functions is used for calculation of group cells coordinates. A local variable is dynamically connected with each cell. The variable name is the number of the corresponding element in a pattern.

General functions (not to be confused with shift functions) can be frequently used for coordinate calculation of group cells. A set of such functions is called a functional pattern. A pattern name consists of prefix `FUN`, followed by a positive integer.

A zone consists of a cellular array name (with the indication of dimension), a pattern name, and a zone body. The zone body is actually a sample. This body is constituted by the list of components, each of which consists of an alphabet symbol (or its code) and a variable, which is a pattern element number.

In the case when a functional transformation is used for a cell state calculation this transformation name is placed in a component instead of an alphabet symbol, and a record of a certain zone is located in brackets after the name. Such a zone body contains only a list of variables. These variables are the arguments of functional transformation. The name of functional transformation consists of prefix MAP and a positive integer.

A microcommand of PML defines a procedure of the unified access to states of group cells and unified processing of states of group cells. The substitution microcommand record consists of two parts: left- and -right, which are divided by the symbol $\rightarrow$. The left part complete record consists of several zones, divided by the symbol $*$. Some of zones can be omitted in the record. If there are no zones at all, the only symbol $*$ remains. The substitution microcommand right part consists of the only zone, which is named operating, and an attribute list.

The execution of the procedure is performed as follows. Cellular arrays, used in microcommand record, are considered to form composition. It means that all cellular arrays, which belong to a composition, are described in the same system of coordinates, and the values of indexes I, J, K for all cellular arrays from composition vary in coordination. For each cellular array (which we refer to as basic) a duplicated array is created (which we refer to as "double"). The procedure consists of iterative steps. Values of coordinates of group cells in basic cellular array, specified in a record of a zone, are calculated at a separate step for each triplet of coordinates I, J, K and for each zone from the microcommand left part.

Calculations for a zone from the right part are performed in the double of the array, specified in a zone record. After that the presence and absence of coincidence of variables values in a zone body and in a group cell is fixed for each zone from the microcommand substitution left part. If the coincidence takes place for all variables in all zones from the left part, values of variables from this zone body are assigned to cells of the group, constructed for a zone from the microcommand right part.

The imitation of parallel execution of a microcommand or a group of microcommands (in other words, synchronization of their execution) is performed by overwriting the states of the cells doubles at the appropriate cells of basic cellular arrays. The indication to overwrite execution just after application of the given microcommand is given with the help of an attribute W, specified in the attributes list. A non-applicability attribute can be specified in the list as well. This attribute is denoted by a symbol L, followed by a name of a cellular array, coordinates of a cell in it and symbol, specifying a state of this cell. A non-applicability attribute indicates that given symbol should be inserted in the cell of the array specified in attribute record if the given microcommand is non-applicable to data.

The cyclic block consists of a name (label), a header, a unlabelled substitutions microcommands list and the operator **END** with the block name. The header begins with a reserved word **TITLE** and contains an information on cellular arrays. The information on each array includes an array name, indexes **I**, **J**, **K**, followed by their range in brackets, and at last a pattern name. The header ends with the list of attributes. The placement of an attribute in the header means that this attribute is the same for all microcommands of the block. If, for example, an attribute W is specified in the header, and other attributes are not present, then the block is executed till non-applicability of all microcommands included in it, and the synchronization is done at every iteration at which all microcommands of the block were once applied. The iteration attribute can be also specified in the header. The iteration attribute is designated by a symbol **R** and a positive integer. This integer defines how many times it is necessary to apply the given block. The iteration attribute can also be written at an isolated microcommand, i.e., microcommand which does not belong to any block.

Except microcommands of substitutions, which make a pithy part of the microprogram, control microcommands are entered in PML. These are required for organization of sequential execution of microcommands of substitutions. The presence of control microcommands requires that some microcommands should be marked by labels. Control microcommands are presented by unconditional and conditional jumps.

The tool microprogram is arranged similarly to the block, it plays an auxiliary role and serves for realization of the most typical, frequently used fragments of computations. The reference to the tool microprogram from any parallel microprogram is done with the help of control microcommands. After execution of the tool microprogram the return is made in a point of a call.

Any parallel microprogram has the following structure. It consists of header, sequence of cyclic blocks and separate microcommands. The header is written down in the same way, as well as the header of the cyclic block, but in addition it contains the information on the sizes of all the cellular arrays, which are used in the parallel microprogram.

When cellular algorithm models are implemented at a computer, the conversion of model source text into PL/1 is carried out with the help of libraries. These libraries contain the sets of unified access and computation procedures for different types zones, used in records of microcommands. Data for model and its functional descriptions are implemented in PL/1. A number of cellular algorithms and structures [1] was described in PML.

2.3. C&M – a language and system of the parallel substitutions simulation

The prototype of C&M [3, 4] is PML. C&M, as well as PML, represents one of the possible PSA notations. C&M derives C language syntax and semantics, but it is filled up by a set of additional data types, data classes, operations and statements. Concept of the alphabet is extended in C&M. A way of using of all data types, permitted in language C, is provided for representation of cell states. Means of sequential control are extended (all C control transfer constructions are available). Nested cyclic blocks are allowed in comparison with PML. Cyclic blocks in C&M language are called synchronous blocks.

In order to include correctly sequential control organization means of C language in C&M, two classes of variables are entered into C&M language. Variables of class OBJECT are intended for use in parallel computations as variable data elements. In other words, these are the cells of cellular arrays. Any change of variables of class OBJECT requires the usage of the statements of synchronization. These means are similar to the attribute list in record of a microprogram, cyclic block or substitution microcommand in PML. Other variables are processed strictly sequentially as it is performed in C.

The synchronous block is the microprogram fragment within the limits of which all statements, varying values of the variables of the class OBJECT (all these statements are reduced to the assignment statement) are carried out simultaneously (synchronously). Until the end of all computations of the synchronous block at an iteration, the modified values of variables of class OBJECT are inaccessible to further use, i.e., the same mechanism of the doubles of cellular arrays as in PML is used. The statements of synchronization (CHANGE, CLOCK, EXHAUST) serve for the description of occurrence of synchronous blocks in the text of the microprogram in language C&M and for organization either once executed, or cyclic computation of synchronous blocks. The CHANGE statement executes its synchronous block single time. The statement CLOCK executes its synchronous block the specified number of times. The statement EXHAUST executes its synchronous block until non-degenerated changes of variables of class OBJECT occur, i.e., up to non-applicability of substitutional statements of the synchronous block. The synchronous block, incapsulated into another synchronous block, is treated separately from its external synchronous block. All changes in its objects are fixed by its own synchronization statement without any dependence on completeness of computation in the external block. The usage of variables of different types (including those of class OBJECT) allows to construct any combinations of sequential and parallel fragments, using variables of class OBJECT for designation of cells of cellular arrays, and using variables of miscellaneous types for composition and auxiliary purposes of the computational process representation.

```
/* C&M model for many integers addition PSA.                          */
                              /* Parameter declarations:              */
#include <sim.h>              /* standard macrodefinitions,           */
#define  num  100             /* number of summands,                  */
#define  dim  64              /* capacity of summands,                */
                              /* Variables declarations:              */
object W[dim][num];           /* summands array,                      */
static area P[5][2] =         /* pattern.                             */
       {0, 0, 0, 1, 1, 0,
        1,-1, 0,-1};

region                        /* Regions of centers' coordinates: */
   R(0:dim-2, 1:num-2),       /* for the first microcommand,          */
   Q(0:dim-2, 1:num-2);       /* for the second microcommand.         */

main()                        /* Addition.                            */
{
   int x, y;                  /* Coordinates of pattern center.       */
   init(W,dim,num);           /* Summands initialization.             */
   exhaust                    /* Exhaust parallel loop.               */
   {
                          /* The first substitution microcommand: */
      on R(x,y)               /* selection of space region,           */
         in Pattern(P,W,x,y)  /* link of pattern with                 */
                              /* cellular array,                      */
                              /* substition microcommand zones.       */
            if(#1 && #2 && ~#3 && ~#4 && ~#5)
               (#1 = 0, #2 = 0, #3 = 1);
                          /* The second substitution microcommand:*/
      on Q(x,y)
         in Pattern(P,W,x,y)
            if(#1 && ~#2 && ~#5) (#1 = 0, #2 = 1);
      pict(W,num,dim);        /* Output.                              */
   }
}
```

Figure 2

In the head part of a microprogram cellular arrays are listed. The type `REGION` specifies the ranges of indexes. The type `AREA` defines a shift pattern in the way similar to C array access.

The substitution microcommand is specified in the synchronous block by set of three statements: `ON`, `IN`, `IF`. The statement `ON` limits the ranges of indexes variation. The statement IN links a pattern with transformed cellular array. The operator `IF` specifies a microcommand body, which consists of two zones named base and operating, written in brackets. Explicitly expressed composition of cellular arrays in C&M language is not implemented.

An example of a parallel microprogram in C&M language is shown in Figure 2. It implements a microprogram of many integers addition algorithm, which was described in Subsection 2.1. There are five elements in a pattern. They are enumerated by numbers from *1* to *5*. These numbers are related to rows in the array **P**. Numbers of pattern elements in bodies of microcommands are marked by the symbol #. In record of bodies of microcommands a number is, actually, variable, which is dynamically assigned to a cell with the same number in the next group, which stands in a queue. These variables are used in expressions and assignment statements placed in base and operating zones of microcommands. **PATTERN** is a procedure of access on a given pattern to the next group cells. The procedure is made in such a way, that the group is built for each pair of values of coordinates x, y when they are modified in given ranges (it is possible to tell, that "continuous" viewing of the array W is executed).

The C&M language was exploited as a source language of a parallel computations simulation system that was implemented in MS-DOS. The converter from C&M language generates intermediate representation of a C&M-program in C language. The intermediate representation is translated into a machine code by a C compiler.

2.4. ALT – a graphic system of parallel microprogramming

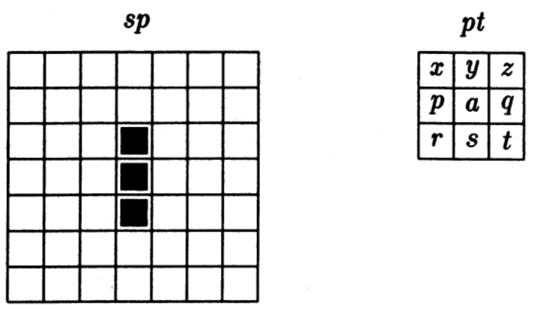
The major problem of research of computing processes with explicitly expressed spatial parallelism by a simulation method consists in necessity of observing simultaneous execution of many interconnected information transformations in large data arrays. An essential drawback of C&M system realization, which is revealed during its test exploitation, was the absence of advanced means of visualization. This fault can be explained by a desire to create a tool which is primarily oriented to program compilation rather than to a dialogue model program execution. It became evident that a system which would have the following features should be created:

- tight integration of the graphic and text forms of the fine-grained model description;
- the visualization of simulation process which is as complete as can be achieved.

Such a system was developed [2, 5]. The ALT system reproduces properties of PSA. Visualization both for designing of a parallel algorithm description and for parallel algorithm modeling are widely used in ALT system. The algorithm description contains graphic representation of objects of that sort of cellular arrays, left-hand and the right-hand parts of substitution microcomands. Visualization gives for parallel algorithm modeling the possibility of supervision of dynamics of separate substitutions application to cellular arrays on a screen.

Unique names are assigned to cellular arrays and patterns. Graphic objects are represented in a window on a screen. This window consists of a number of pages. Only one of them can be viewed at a time. The separate graphic object is composed of colour cells. A colour represents a current state of a cell. Considerable attention has been given to the visualization of 3D objects. 3D object is shown as a pack of layers. The layers form on a screen a dimetric projection so as to ease a 3D perception of an object by a user. One of layers is open. It is shown in the screen as a matrix of colour cells.

The program scheme language and the functional language (subset of C language) are used for writing of parallel microprogram text. A substitution microcommand is represented by operators **IN**, **ON**, **AT**, **DO** in program scheme language. The purpose of the operators **IN** and **ON** is similar to that of these operators of C&M language. The pair **AT**, **DO** replaces the C&M language **IF** operator. Names of the substitution microcommand left-hand part zones are listed after the operator **AT**. Either a name of a substitution microcommand operating zone, or a name of functional transformation which is recorded in the functional language is specified after the operator **DO**. All functional transformations are displayed in a special window. Main distinction of the parallel substitution microcommand representation in ALT system from representation in languages PML and C&M lies in the fact that the names of zones are used in program scheme language only, and



```
ex
    in sp
        at pt
            do lf
```

```
lf()
{
    (a && (x+y+z+p+q+r+s+t !=3) && (x+y+z+p+g+r+s+t !=2)
    ||!a && (x+y+z+p+q+r+s+t==3))?a=!a;
}
```

Figure 3

each zone has a graphic image. Synchronization operators in the program scheme language are similar to those in C&M, but usually their names are recorded in reduced notation, for example CH, CL, EX. A parallel microprogram scheme is also displayed in its own window. The Conwey's "Life" model which demonstrates the basic features of ALT and its language is shown in Figure 3.

A number of fine-grained algorithms and structures [2, 9] was developed in ALT. Cellular technology of parallel computations organization, which is based on ALT [2, 6, 9], has been formed in an essential degree.

3. WinALT – a new simulation system of computations with spatial parallelism

System ALT operating experience has shown vitality of through mutual visualization of dynamic and static elements of model, while both of arrays and graphic images of commands which describe data transformations are simultaneously presented in the screen space.

At the same time, construction of a rather wide set of fine-grained models algorithms and structures was also shown, that the user needs such a system that can provide both a possibility of the work within an interactive environment with the detail analysis (and visualization) of evolution of computation process in each cell and the possibility of execution of parallel information transformations during many steps in large data arrays.

It has also become evident that in connection with diverse and rather frequently varying (depending on a class of simulated structures) user's requests which can hardly be satisfied by a creation of a set of standard means it is appropriate to provide the system with the possibility of its replenishment by means, developed by the user himself, possibility of configuration and programming of system according to user's specific interests.

The system ALT does not meet the requirements mentioned above because of its limitations, which are most pronounced at employment of system for imitation of neural computations and for designing of complex fine-grained computing structures from sets of more simple structures.

The basic limitation of ALT system lies in the fact that the interpreters of program scheme and functional languages (subset of C language) are the independent programs, which cannot communicate with the modules produced by a standard C compiler. Therefore the ALT system functionality cannot be extended by these modules. Such a drawback strongly limits representational possibilities of language means of ALT system.

Other limitation of ALT system is the absence in its interpreter of a fast run mode, which would allow to simulate work of fine-grained algorithms with large volumes of data within acceptable time on the computer. For example the fast run mode would allow to use ALT system not only for research and debugging of fine-grained algorithm, but also for production of real results of its work.

Accumulated experience also shows that a vital need is felt to bring the interface of system in conformity with modern demands of a user for comfortable work in application area, i.e., wide and convenient set of tools for editing textual and graphic parts of model is required, it is necessary to use as much as possible completely the potential of operating system.

Simulation system (WinALT) [10] satisfying the increasing requirements of users, which are involved in researches of fine-grained computations was developed under the direction of the author of this article. The new system retains the functionality of ALT. It also has a lot of additional functions, which expand both the user interface, and its modeling functions. It is possible to say that the new system has combined the best features of two previous systems. In addition WinALT implementation is done on the modern Win32 (Windows 95, NT) platform. The usage of this platform gives advantages in GUI design, allows to exploit its rich functionality in multiprocess organization and synchronization, memory management.

The WinALT GUI is based on the usage of standard Microsoft GUI means such as document windows, menus, toolbars and dialog windows. One of the three implemented toolbars is standard for any Windows application and supports a set of buttons for file loading, editing, saving and printing. This WinALT toolbar has a lot of useful GUI features which are absent in ALT. Two other toolbars are specific for WinALT. They are "WORK" and "TOOLS". These toolbars support means of visualization, intended for both designing and modeling of parallel algorithms in a way similar to that of ALT. The image of a graphic object is similar to that in ALT. As in ALT the program of a model is displayed in a separate window. The number of tools for graphic object transformation is increased in comparison with ALT. For example, the "TOOLS" toolbar contains buttons for initialization of cellular arrays with random states, importation of objects from the BMP format. The "WORK" toolbar contains buttons for graphic object size and dimension modification, 3D object layer editing, cell block transfers within objects. WinALT breaks the limits for object size imposed by an obsolete MS-DOS environment which was the platform for ALT. Object sizes are limited only by amount of available virtual memory. Model text editing support tools, modular construction and model program debugger are available.

The main goal of WinALT language is to maintain extremely flexible structurization of fine-grained computation expression devices at different detalization levels. Another goal was to construct a system which can be easily extended by new functionality. Just as the ALT program scheme language the WinALT language has its own interpreter and does not depend on any external tools. But its set of tools for designing of the program models forms a superset of C&M language, which was constructed as extension of C language. The language can be divided into three levels in accordance with types of its syntactical constructions.

The first level of WinALT language is represented by a set of syntactical constructions which is habitual for the most structured programming languages. Namely, there are conditional loop, **IF-THEN** statement, function and procedure support and immediately executed assignment for variables or cells. There is the set of standard unary and binary arithmetic and logic operations in the language. Cellular array processing looks more like a work with ordinary arrays at this level.

The second level is based on the first and is formed by means which are intended for compact representation of distributed in space parallel computations. The means are identical with those in C&M and ALT systems. As in C&M language WinALT variables are divided into two classes: ordinary variables (local or global) and objects. This level is represented by synchronous blocks and loops (operators: **EXHAUST**, **CLOCK**, **CHANGE** are used), substitution describers: **IN**, **ON**, **AT**, **DO**, and synchronized assignment operator.

The third level is formed by language constructions, which enable integration of modules, written in C or C++ into WinALT. This modules are dynamic linked libraries with a specific set of interface functions and named ACL, which stands for ALT C language Library.

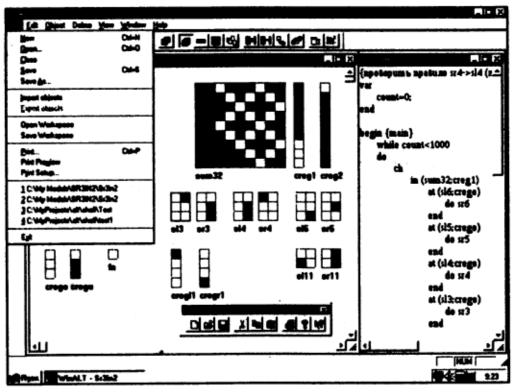
While all the ordinary functions are to be written in the usual way, all the exportable functions which are intended to be visible from a WinALT program need to be specially declared and defined.

A declaration of an exportable function is placed into **DECLARE_ACL** section. Each declared function should be defined in **ACL_EXPORT** section. A definition consists of its C name, WinALT name and a quantity of parameters to get.

The last and the most important thing to do is to implement a function. An interface was designed to allow an ACL to access cellular arrays, change state of WinALT program execution and control other parameters in WinALT system.

The program of model consists of the main program block, placed between **BEGIN** and **END** keywords. ACL module function calls, declarations of global variables, descriptions of functions and procedures could precede this block.

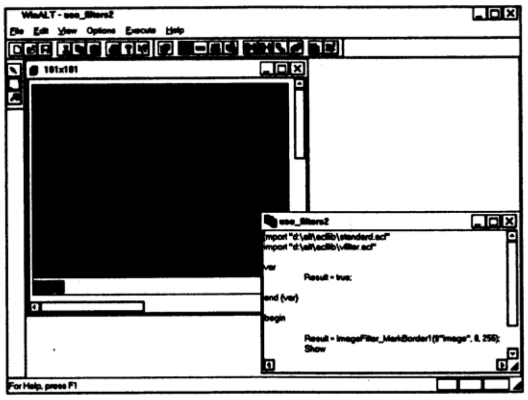
A screenshot of the WinALT is shown in Figure 4. Cellular arrays, left and the right parts of substitutions for the model of 3$\rightarrow$2 adder [2] are shown in an object view window, a fragment of the simulating program is represented in the program scheme edit window. As a whole, the model has a style of ALT system models.



```
{проверить правило sr4->sl4 (z
var
    count=0;
end

begin {main}
    while count<1000
    do
        ch
            in (sum32:creg1)
                at (sl6:crego)
                    do sr6
                end
                at (sl5:crego)
                    do sr5
                end
                at (sl4:crego)
                    do sr4
                end
                at (sl3:crego)
                    do sr3
                end
```

Figure 4



```
import "d:\alt\acllib\standard.acl"
import "d:\alt\acllib\vfilter.acl"

var
        Result = true;

end (var)

begin

        Result = ImageFilter_MarkBorder1(@"image", 0, 255);
        Show
```

Figure 5

The local filter model for picture processing is shown in Figure 5. The ACL module external functions are used in this model.

4. Conclusion

WinALT language and auxiliary tools allow easily present a great number of interconnected processes with different complexity in a model program, which imitates a fine-grained algorithm or a structure. These processes can have different types of interconnection and distribution in cellular data space.

WinALT functionality can be changed and extended by a user.

WinALT provides as much as possible wide sphere of tool requirements of the researcher of large discrete dynamic distributed systems, not removing his for limits of paradigm of cellular model of parallel computations, which is very natural for the given area of researches. A conclusion can be given that

WinALT provides a set of tools for miscellaneous demands of researchers in exploration of large discrete distributed dynamic systems (e.g., in physics, biology or technics) without dependence on a narrow specialization.

References

[1] P. Anishev, S. Achasova, O. Bandman, S. Piskunov, S. Sergeev, *Parallel Microprogramming Methods*, Nauka, Novosibirsk, 1981 (in Russian).

[2] S. Achasova, O. Bandman, V. Markova, S. Piskunov, *Parallel Substitution Algorithm. Theory and Application*, World Scientific, Singapore, 1994.

[3] S. Piskunov, Yu. Pogudin, *Parrallel microprogramming as a tool for microprocessor systems design*, Proc. Int. Conf. SAPR SVT'89, section 2, Leningrad, April 17–21, 1989, 197–205 (in Russian).

[4] Yu. Pogudin, *C&M – C language extension for microprogram applications*, Informatics and Programming, Computer Center, Novosibirsk, 1989, 62–85 (in Russian).

[5] Yu. Pogudin, *ALT – a Grafical System for Parallel Microprogramming*, Parallel Algorithm and Structures, Computer Center, Novosibirsk, 1991, 77–88 (in Russian).

[6] V. Markova, S. Piskunov, Yu. Pogudin, *Formal methods and tools for design of cellular algorithms and architectures*, Programmirovanie, **4**, 1996, 24–36 (in Russian).

[7] K. Brenner, A. Huang, N. Streibl, *Digital optical computing with symbolic substitutions*, Applied Optics, **25**, No. 18, 3054–3060, 1986.

[8] Yu. Kornev, S. Piskunov, S. Sergeev, *P. N436350 (USSR), Binary adder*, Byulleten' izobretenii, No. 26, 1974 (in Russian).

[9] V. Markova, S. Piskunov, *Computer models of 3D cellular structures*, Lecture Notes in Computer Science, **964**, 70–84, 1995.

[10] D. Beletkov, M. Ostapkevich, I. Zhileev, *A simulation system of computation with spatial parallelism*, Proc. Conf. of Young Scientists, Computer Center, Novosibirsk, 1997, 16–23 (in Russian).