Bull. Nov. Comp. Center, Comp. Science, 17 (2002), 89–103

Imitational simulation of fine-grain algorithms and structures*

M.B. Ostapkevich, S.V. Piskunov

The WinALT system for imitational simulation of fine-grain structures and algorithms is presented in the paper. Its architecture is substantiated; user's interface is described, possible applications are outlined. It is demonstrated that open architecture of the system allows to construct different versions of system for sequential and parallel computers and supports models of nearly every kind of fine-grain parallelism with large data amount.

1. Introduction

One of main kinds of parallelism that lies in the foundation of parallel computation technology is called fine-grain.

Its attractiveness can be easily justified. Many applications have extreme (natural) parallelization only within this kind of parallelism. Among such applications signal and image processing, mathematical physics models and graph problems, custom processor architecture design for massive information processing can be mentioned.

Such stages of research as observation of physical phenomena evolution, verification or complexity (time, hardware, connection topology) estimation for parallel computation structures by their algorithmic descriptions cannot be accomplished without computer aided simulation. This makes actual the development of fine-grain algorithm simulating tool. Such tools were developed in SSD ICM&MG for the last several years in the form of open system of imitational simulation WinALT. In this article, which is a sort of an overview, the development of system architecture is considered for both sequential and parallel versions of the system. The user's interface and the area of its application are outlined.

2. The WinALT basic version [1, 2]

2.1. System design guidelines

The main requirement that is demanded from the system is universality within fine-grain class of parallel algorithms.

*Supported by the Russian Foundation for Basic Research under Grant 99-07-90422.

Such universality makes this system unique in a certain sense among known systems [3, 4]. Actually, all these systems support only one kind of cellular architecture. The existence of only one system (with one fine-grain model description language) for all researchers makes it possible 1) to compare results of different developers; 2) to improve the system in cooperation with numerous users; and 3) to create libraries of models and construct new models as combinations of existing ones.

The decision of single tool construction predefined the following:

1. A formal ground must be selected, which would integrate in itself all features of particular fine-grain computations.
2. An architectural principle of system construction ought to be chosen so that a system would be easily adapted for growing number of simulating phenomena and ever for increasing user demands for tools and functions included into system and performance of hardware hosting the system.

Fine-grain computational model named Parallel Substitution Algorithm (PSA) [5] was chosen as a formal ground of the system. The PSA combines substitutional character of Markov's algorithm [6] with spatial parallelism of cellular automata [7]. The PSA is based on common for these notations associative mechanism of operation application. The PSA represents natural parallelism of computations, which means that at each step all the allowed actions are executed for all the available data. Such constitution of a model allows to describe within it cellular automata, neural and cellular-neural networks, systolic structures, homogenous structures with programmed logic and so on.

From the architectural point of view, the WinALT is built as an open system [8]. It was created quite universal, but in dynamic rather than static way when all useful functions are embedded. A user has means to add new modules with a certain unified interface. He can replace or exclude modules and modify their relationships. These modules can be produced by a user himself or be parts of standard libraries [9]. The openness of the WinALT [9] is the main difference from its ancestor, ALT [10]. An open system [11] is characterized by such features as extensibility, scalability, portability, interoperability, and friendliness of user's interface.

2.2. The WinALT user's interface

The interface represents main PSA properties, which are concluded in the three following statements:

- The processed information is presented as a cellular array, which is a set of cells, entities of atomic type (bit, character, number et al.) with certain location within array.

- Algorithm is presented by set of parallel substitutions. A substitution has left and right parts. Left part expression generates an associated cellular array for each cell name. If this array exists in processed array, then the substitution is executed. Execution means the replacement of a certain basic part of found array by the right part of substitution of the same cell name.
- The process of computation is iterative: all the applicable in a cellular array substitutions are executed at each step. The execution is finished when there are no more applicable substitutions in processed array. It is this array that is the result of the PSA work.

The writing of correct parallel algorithm is not an easy task. To facilitate it, the local information transformations are used in the PSA in graphical form of certain cellular object discrete space of cellular arrays, left and right parts of substitutions. Left and right parts are represented by cellular arrays that called templates. It is this approach that is the principle of the interface construction. Its property is tight integration of textual and graphical forms of model description. The interface is divided into graphical and language parts. The language part of model is called simulating program.

2.2.1. Graphical interface. When conducting simulating process that consists of multitude of data parallel transformations, it is very important to present the results in an easily readable form for a user, so that he at least could understand the qualitative picture of processes in model. Thus the friendliness is the key feature of a GUI.

Multiwindow interface is well known to a Windows user. The tools and functions of the system are gathered into the GUI elements, such as panels and menues [12].

The representation of cellular object: the array or the template is the main part of graphical interface. A 3D object is visualized as a deck of rectangular layers, one of which is visible at the screen. Also, an object can be shown as evolvent of layers in a plane. A layer is a matrix of the colored cells. The colour denotes the cell value.

A model forms a project, which includes a set of cellular objects, all the simulating program sources, and all the used external libraries. All the parts of a project and its tree can be shown in their respective windows.

The system tools and services gives a user a comfortable environment for the construction and the modification of cellular arrays and simulating programs. The environment includes means for data transformations observations. In debug mode, it is possible to trace all substitution applications.

The model window, given in the next section sample, gives an impression on graphical interface.

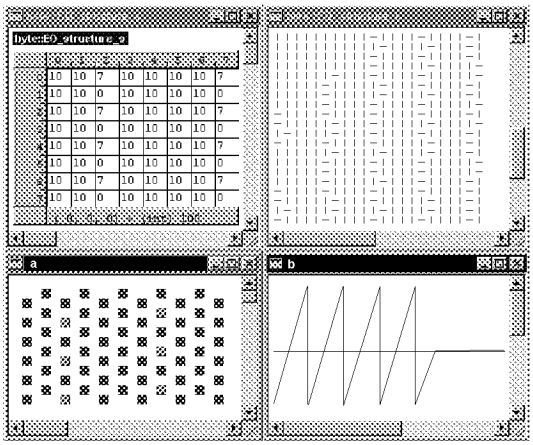

Figure 1. Sample the OVE screenshot

Of course, the list of implemented functions given above is by no way complete. As the WinALT is an open system, it can be enriched by subsystems with new functions for graphical representation of objects in models. One of such subsystems that were included in the system is OVE [13]. It enables the inclusion of custom visualization modes in the system. These modes are implemented by external libraries written by a user. They are called object visualization drivers. The samples of their screenshots are shown in Figure 1 for square and hexagonal grids, value visualization by color, arrows and numbers.

2.2.2. Language interface. The language is used for the model textual form writing. It is discussed and substantiated in details in [14, 15]. Extensibility is one of distinctive properties of the language. The language consists of three parts. The first one contains operators for compact representation of parallel computations spread in space and based upon the PSA. Another part unites statements of general purpose sequential language for structured programming. The last part is presented by means of library construction in the system language and function import from libraries included into widely spread general purpose languages, C or C++ in the current version.

The first part of the language is central. A lot of its operators is similar to the ALT operators. A parallel substitution is set by a compound parallel operator that includes `in`, `at`, `do` operators. The compound operator defines a ubiquitous execution of local information transformations within a cellular array.

The `in` operator with a cellular array name defines the space of substitution application. The substitution itself is set by `at-do` tuple of operators. The `at` contains the name of left-hand template, while the `do` contains the name of right-hand template. Just as in the ALT system, a vector form of

substitution exists. Such substitution makes a transformation in a group of objects at once. In this case, each tuple operator is followed by the list of cellular object names in round brackets. Placing templates into lists means that their movement in cellular arrays is performed in coordination. The information transformations are also performed in coordination.

The iterative procedure of the PSA application is implemented by `ex-end` synchroblock. For the sake of convenience, there are two more types of synchroblocks: `clock-end`, `change-end`. The `clock` executes a specified number of iterations. The `change` is executed once. The number of iterations in `clock` can be set by a constant or expression. Also, expressions can be used instead of constants in `on` operator, that limits the area of substitution applicability in cellular array by its certain part.

The same situation is with `step` operator that sets applicability step. The `let` operator, which implements synchronous assignment, is among new operators. Included into a synchroblock, it changes the value of its parameter, a variable, only at the end of synchroblock iteration. This operator allows to imitate substitutions with any naming functions [5] and not just template description functions.

The existence of the second part in the language is justified by several reasons. First, many parallel algorithms have at least a small part of sequential computations [16]. Second, there is a need to describe complex data transformations by the functional substitutions. Such transformations are formed as procedures and functions. Their names are used as a parameter in `do` operator. Third, it is required to describe auxiliary procedures, such as data input/output.

The unification of parallel and sequential parts of the language is reached by the ability of synchroblocks to contain both types of operators. Nevertheless, the semantics of sequential operators is not altered in synchroblocks, the results of their execution are assigned immediately.

The third part of the language is responsible for extensibility. It contains `import` and `use` operators. These operators import external libraries.

Pascal was chosen as syntactic prototype of the WinALT language. This choice means that the structurization of model program, procedure and function syntax and most of keywords are borrowed from Pascal.

The ALT and the WinALT systems are used for the design and the research of computing devices models [5, 17]. The use of the WinALT for cellular-neural networks is presented in [18]. The technology of model construction is considered in [19]. Here we shall introduce the model of physical process.

Example of model description (Figure 2). The model implements circumfluence of obstacle by fluid stream. It is based upon [20, 21] and is called the FHP model on hexagonal grid.

There are six types of particles with six directions of motion. The angle

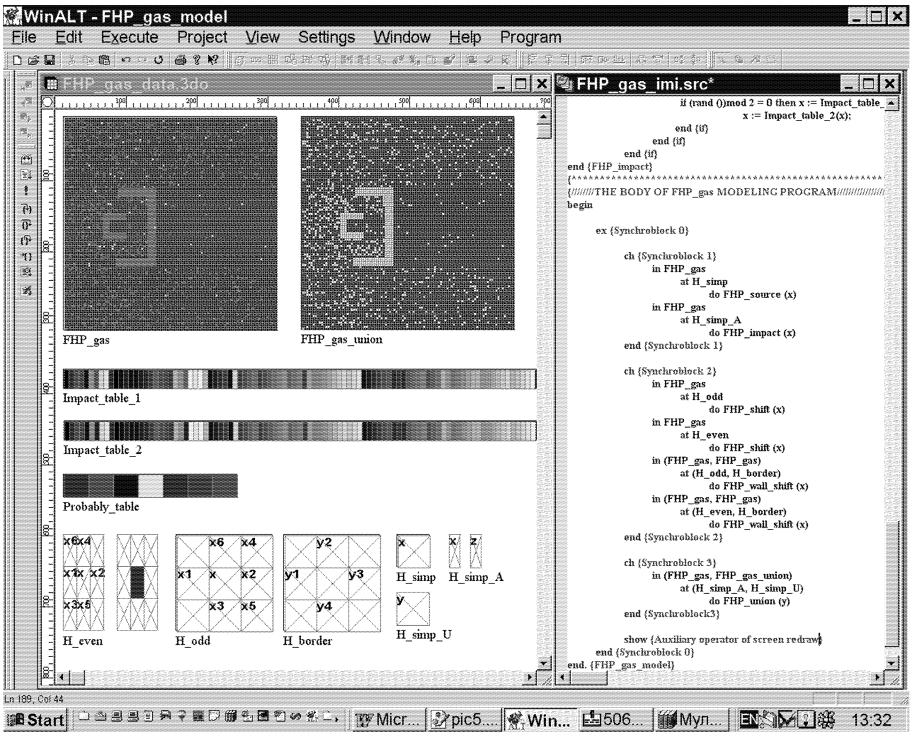

Figure 2. Sample model screenshot

between neighbor directions is 60 degrees. There is also a motionless particle. The grid is represented by two layer `FHP_gas` cellular array. The layer zero is a working field for moving particles. Digits from 0 to 6 of cell state in this layer correspond to seven kinds of particles. The seventh digit corresponds to the obstacle. The first layer is used for hexagonal grid imitation. This imitation is done by separation of layer into even and odd rows and by two two-layer templates `H_even`, `H_odd`. In even rows of `FHP_gas`, the cells are set to 1, while in odd rows – to 0. The central cell in the first layer of `H_even` is set to 1, while the same cell in `H_odd` is reset to 0. The proposed way of cell state setting enables associative mechanism of template alternation in the simulating process. The variables written into cells of the layer zero in `H_even`, `H_odd` templates sets the direction of particle motion: `x1` (goes right), `x2` (goes left), `x3` (goes up right), `x4` (goes down left), `x5` (goes up left), `x6` (goes down right). The `x` variable is introduced for the case of motionless particle. The cells of the first layer (left most column) represents a line of particle sources. `FHP_gas_union` is entirely auxiliary. It serves for the convenience of circumfluence observation by a user.

The simulating program is made as follows: three synchroblocks 1, 2, 3 (`ch-end` couple) are included into synchroblock 0 (`ex-end` couple). Such

program structure means that the update of cell values is done within a loop consisting of three steps. An essential part of the program is depicted in the right window in Figure 2. The use of functions in `do` operators is typical for this program. These functions transform values of local variables that correspond to template cells.

The update of grid nodes states is done within two steps of iteration. The first step (synchroblock 1) performs the impact of particles. Impacts are taken from [21] and described in `Impact_table_1`, `Impact_table_2` arrays. They are executed by `FHP_impact_table` function and `H_simp_A` template. Unlike [20], this model as in [21] has a possibility of particle deviation from its direction. Deviations are set in `Probably_table` array and performed by the same function and template as in impact. The generation of particles is also done at the first step by the `FHP_source` function and the `H_simp` template. The second step (synchroblock 2), each particle is moved along its direction from its current position to adjacent. This motion is performed by the `FHP_shift` function `H_even`, the `H_odd` templates for even and odd `FHP_gas` array rows, respectively. At the same step, the bouncing from wall is done by the `FHP_wall_shift` function. This action is done by the vector operators `in-at-do`. The vector components are listed in round braces after each of operators. For even and odd strings of the `FHP_gas` array the same `H_even`, `H_odd` templates with addition of `H_border` are used. The latter allows to recognize an obstacle.

At the third step (synchroblock 3), also, a vector command of substitution moves the `FHP_gas` cell states to `FHP_gas_union`. The function `FHP_union` and the `H_simp_A`, `H_simp_U` templates serve for that.

To give an impression about functions used in the model, one of them is shown below. It is defined for all cells of layer zero in `FHP_gas` and it operates `x`, `z` variables of the two layer template `H_simp_A`. The source constant is used to locate source cells in layer zero. `Mask_x0` is for separation of the motionless particles. The `wall` constant serves for the wall detection. The `rand()` is a library function for the pseudo-random value generation. The `rand_tran` is a constant that sets the probability of single particle mutation. `r` is a stack variable. The first `if` operator cuts source cells. The second locates single cells, while the third changes the particle directions. The `else` operator marks cell states, which are different from single cell states. The fourth `if` with probability 0.5 changes cell state to new, in this case, the old cell value is used as address in the `Impact_table_1`, `Impact_table_2` arrays. These arrays code a transition table of a certain stochastic automaton. First 128 positions code configurations of cell impact. Last 128 entries serve for coding of cell bouncing from an obstacle.

```
function FHP_impact;
   r
begin
```

```
   if z <>source then
      if (x=mask_x0) or ((x<>0) and ((x)mod 2=0) and (x<wall)) then
         r := (rand ())mod rand_tran;
         if r < 7 then
            x := Probably_table(r);
         end {if}
      else
         if (rand()) mod 2=0 then x := Impact_table_1(x);
         else
            x := Impact_table_2(x);
         end {if}
      end {if}
   end {if}
end {FHP_impact}
```

The sample explicitly demonstrates the compactness and intuitive understandability of graphical model representation and readability and conciseness of language means. The main block of model program is entirely presented at screenshot. The part of program that is not depicted at the screen and contains library function calls, variable and template declarations, procedures and functions, which are used in the main block, is written in a standard Pascal-like form.

2.3. Architecture of basic version of system

2.3.1. General description. The architecture is described in [1, 9]. There are three distinctively discernible parts of the system: kernel, graphical, and language subsystems. The system extensibility and scalability is obtained through modular design, which permits the implementation of most functions in external libraries. Operations for the support of such libraries are inside the kernel. The main means of intermodular interactions is event. This allows to minimize the number of interfaces and ease porting to distributed environments.

2.3.2. Kernel. Main kernel modules are the following: object manager and external library manager. The destination of modules and their interaction with other modules will be described below in the overview of main subsystems. Object manager implements operations for cellular object creation, modification of object properties and cells. The manager supports cellular objects in different formats. In fact, the real implementation of format support is made outside the manager, in external libraries that interact with manager through its interfaces.

Library manager solves the problem of language extensibility by giving a mechanism of language enriching with functions that are implemented in external dynamically linked libraries.

2.3.3. Graphical subsystem. Graphical subsystem is module built as a Windows application. It was developed in Microsoft Visual C++ with the MFC library. Basic and derived classes allow it to act as object editor and set of tools. The separation of physical, logical and visual levels in the WinALT system allows graphical subsystem to use cellular objects in a uniform way not depending on which particular format they are created in. The interaction between layers is done via kernel interface.

2.3.4. Language subsystem. It implements the interpreter of system language. The preliminary translation into internal code is used because the performance of interpretation is a critical parameter. The kernel interfaces for object and external library processing are used by a translator. The subsystem exists in two forms: 1) coupled with graphical subsystem; 2) stand-alone console version. The console version does not have any tools for object visualization and editing. It is capable only to compile and execute programs.

2.3.5. The development of the WinALT architecture. In the process of the WinALT development, the modification of its architecture was planned and is under implementation now. It increases the degree of the system structurization. The essence of modification is the introduction of new layer in hierarchy and general purpose function concentration in a library, which is external for the WinALT. The library is called Dynamically Configurable Modular System (DCMS) [22]. This library can be considered as a part of software platform upon which the system is implemented. It enriches the platform with new interfaces that contain operations for a) associative data access by a string key, b) the DCMS format typed value management, c) event-driven intermodular interface.

Associative search by a unique key is used in language subsystem in translation block and in object manager for object descriptor retrieval by identifier. It is also required for storage of substitution set indexed by names and coordinates in substitution storage manager. Associative search is used for sparse array representation as well. The data structures for associative search [23] were developed. Their efficiency in comparison with binary and bitcoded trees was shown for the data search in fine-grain algorithms simulating system. The DCMS from a virtual machine point of view has operations for value processing. These values always have certain type. They are represented in single format in the RAM and their operations are contained in the so called value manager. The DCMS values are used as unified form of intermodular data exchange. For example, interaction between topology manager and console or graphical environment entirely relies upon the DCMS values. The DCMS event-driven intermodular communication is also widely used because it simplifies modifications in system even at interface

level.

In the conclusion of Section 2 it should be mentioned that the size of system source texts is 3MB.

3. Description of cluster version

Sufficiently detailed simulation of cellular algorithms for non-linear dynamics, image reconstruction, imitation of parallel computation structures require a huge amount of computational resources. For example, the experiments of such types were held on cellular arrays with sizes from 1024×1024 up to 5000×5000 on Connection Machine [20]. The fields of even greater sizes are required in the 3D case. That requires porting of the system to parallel computer. Further the architecture of cluster version and construction of models in it shall be considered.

3.1. Architecture of cluster version

Cluster version is an extension of the basic WinALT version. It consists of three parts: client, daemon, and server.

Client part is represented either by graphical or console version of the WinALT. Server is a modified console version. Daemon is utility, which is currently implemented only under Win32. The user interface for its set up consists of icon in Explorer task bar and a menu associated with it. Both server and daemon are installed on each cluster node.

The interaction of components goes as follows. Daemon is activated at startup on cluster nodes and is waiting for commands. A user launches client part. It requests daemon to start server parts on each node.

The parts of cluster version have some cluster specific modules. The main ones are:

- network communication module, which allows to make connections between machines, perform synchronization and data transfers; there are two versions of this module: one for client and daemon and the other for server;
- modification of object manager for client version; it enables object division for spread on cluster hosts;
- server's modified module of syntax analyzer and code generation including generation of communication code;
- server's modified interpreter of internal code including cluster data transfer and synchronization procedures.

The separation of network communication module allows to utilize network communications from different libraries (e.g., PVM [24], MPI [25],

sockets [26]). That can be accomplished by creation of specific versions of this module.

The choice between low and high level communication tools is resolved by balance of efficiency and flexibility on one hand and ease of implementation on the other. In the WinALT, as a simulating system, the important of the two former factors is high. That has leaded to refusal in usage of the PVM and the MPI at least at this stage of development. Current version relies upon sockets library as a means of interaction between part of the distributed application both at initialization and computation phases. Sockets is currently a *de facto* standard interface of TCP/IP [27].

The source code size of cluster specific modules is 243KB.

3.2. Model construction and execution on cluster

3.2.1. Porting model to cluster. Model that was built in the basic version of system is automatically ported to cluster if it meets certain limitations:

- data is represented by one or several cellular arrays that have equal sizes at least for dimension, which is used for division of these arrays into parts that are to be spread among cluster machines;
- only local data transformations set by templates with sizes independent from processed data array sizes and at least one degree less than the latter are used;
- only fully defined local data transformations are used;
- local transformations with associative search are used.

Such limitations are by no means burdensome as they are actually the properties of most fine-grain algorithms and structures from classes (see Section 2.1) which are targets of designed system. Typical sample from Section 2.2.2 proves this thesis.

Automatic porting of model from one version of system to another means that a user is relieved from explicit coding of communications. The execution of a model is initiated by a user on the host where the client is running. Data preparation and their transmission to cluster nodes is performed by interacting client and daemons without user participation.

The phase of user construction in the basic WinALT version can be considered as debugging one, when a user has a possibility to use all the available model construction and debugger functions. Later the results of single machine simulations can be used as reference ones in first (test) launch on the cluster.

It is more complicated when a model in basic version uses some types of operators for simulation acceleration on single machine. Among these

operators are those Pascal-like from the second part of the WinALT language if they are used for control constructions in the main block of simulating program.

In this case before porting, a program has to be manually modified. Otherwise the parallel simulation might loose its gain in performance over sequential. First, remaining within limitations of the WinALT language, "undesirable" operators should be eliminated if that is possible. Additional layers in templates or extra templates can be used for this. Usually such modification made with the help of visualization tools is not difficult.

Nevertheless, if a model is radically changed and data amount increases considerably, a user has to program communications manually. The minimal set of functions exists in cluster version for this purpose. In this case, the results of simulation in sequential mode should be used for comparison and verification.

3.2.2. Model execution on cluster. To execute a model on cluster, a user either runs a) console version from command line, e.g., from Far Commander being in model project directory, or b) graphical version from explorer menu `start/Programs/WinALT/WinALT`. In either case, he specifies the obligatory topology parameters, name of main source program to simulate, arrays to split, arrays to send back to a client as results of simulation. The templates are always sent identical to all cluster nodes.

To make a transition of model from 2.2.1. to a model running on two hosts, it is enough to issue a command line:

```
m:\winalt\bin\xaltcon.exe -tFHP1 -x2 -y1 -z1 -h(0,0,0)192.168.76.1
-h(1,0,0)192.168.76.3 -rFHP_gas -rFHP_gas_union FHP_gas_imi.src
```

The meaning of parameters is explained in the following list:

`winalt` – root directory of the system;
`-tFHP1` – sets topology name;
`-x2` – sets size by x to two;
`-y1` – sets size by y to one;
`-z1` – sets size by z to one;
`-h(0,0,0)192.168.76.1` – sets IP for host in grid position (0,0,0) to 192.168.76.1;
`-h(1,0,0)192.168.76.3` – sets IP for host in grid position (1,0,0) to 192.168.76.3;
`-rFHP_gas` – declares object `FHP_gas` as divided and sent back as result;
`-rFHP_gas_union` – declares `FHP_gas_union` as divided and sent back as result;
`FHP_gas_imi.src` – sets simulating source name.

After initialization the client part compiles simulating program for two purposes: a) verifies syntax correctness; b) builds list of used cellular objects and included files. If the program contains errors, the execution is terminated. Otherwise, the client part prepares data for each node by making copies of some files and splitting some of objects. Then it gets daemons of cluster hosts into exclusive usage and sends prepared data to them. They keep it in their local file systems.

After completion of these steps, the client initiates simulating program execution on cluster nodes through a command send to each server. Server parts simulate the model program and at the end of each synchroblock the synchronize data in divided arrays. Client polls the status of server processes and retrieves the console output. After termination of server parts, the arrays declared as results are sent back from daemons to client. Client part pastes them back into single objects.

3.3. Results of the WinALT parallel version testing

The testing of cluster version was done for cellular automata models with local neighborhood of five or nine cells and two types of source data initialization: data spread over all array or concentrated in central part of array. The number of machines with equal performance varied from two to nine. Cluster version has shown good speedup rate [28] which is proximate to the number of hosts for models with steady data distribution (changes from 2.92 for three hosts to 8.21 for nine) in the case of main array size 201x201. In the case of initial data concentrated in center, it is 2.61, 3.51, 3.77 for 3, 4, 5 machines, respectively. The degradation of speedup rate is explained by the fact that the amount of data along the axis of object division is diminished in a single host while template size, which determines the size of data sent between hosts, remain constant. That means that the communication time augments while computation time decreases.

4. Conclusion

The system gives comfortable environment for accomplishing of all simulation steps: from model construction to simulating with particular source data sets on single or multiple processors. The means of system allow to create concise and rather self-documented model description of models for all known fine-grain kinds.

Further system development is planned in two directions.

The first direction is related to increasing of its portability. Currently only console edition can work both in Win32 and Linux. It is planned to create portable graphical subsystem (its prototype is described in [13]) and then porting parallel version to Linux.

The second direction is dedicated to increasing of system efficiency for fine-grain models with features, such as concentration of data at a certain part of array. In parallel version, it is planned to implement dynamic load balancing for host load.

References

[1] Beletkov D.T., Ostapkevich M.B., Piskunov S.V., Zhileev I.V. WinALT, a software tool for fine-grain algorithms and structures synthesis and simulation // Lect. Notes in Comput. Sci. – 1999. – Vol. 1662. – P. 491–496.

[2] Beletkov D.T., Ostapkevich M.B., Piskunov S.V., Zhileev I.V. The tools of language and graphical interface of a simulating system for computations with spatial parallelism // Proc. of the VI-th Intern. Workshop DDP. – Novosibirsk, 1998. – P. 228–232 (in Russian).

[3] Cellular Automata Simulation System. – `http://www.cs.runet.edu/~dana/ca/cellular.html`.

[4] CellLab. – `http://mathcs.sjsu.edu/faculty/rucker/cellab.htm`.

[5] Achasova S.M., Bandman O.L., Markova V.P., Piskunov S.V. Parallel Substitution Algorithm. Theory and Application. – Singapore: World Scientific, 1994.

[6] Markov A.A. Theory of Algorithms // Proc. of Mathematical Institute of Academy of Science of USSR. – 1954. – Vol. 42 (in Russian).

[7] Codd E.F. Cellular Automata. – New York, London: Academic Press, 1968.

[8] Wirth N. A plea for lean software // IEEE Computer. – 1995. – Vol. 28, Iss. 2. – P. 64–68.

[9] Ostapkevich M. The open architecture of WinALT // NCC Bulletin, Series Comp. Comp. Science. – Novosibirsk: NCC Publisher, 1998. – Iss. 9. – P. 93–106.

[10] Markova V.P., Piskunov S.V., Pogudin Y.M. Formal methods, language and instrumental tools of cellular algorithms and structures synthesis // Programmirovanie. – 1996. – № 4. – P. 24–36.

[11] Filinov E. The selection and development of the open system environment conceptual model // Open Systems. – 1995. – Vol. 6, № 4. – P. 32–46.

[12] Beletkov D.T. Graphical construction of computer models of 3D cellular algorithms and structures // Proc. of Young Scientists Conf. – Novosibirsk: ICMMG, 1998. – P. 3–13.

[13] Ostapkevich M., Shashkov D. The construction of graphical interfaces // NCC Bulletin, Series Comp. Comp. Science. – Novosibirsk: NCC Publisher, 2001. – Iss. 14. – P. 59–64.

[14] Ostapkevich M. Language tools of WinALT system // Proc. of Young Scientist Conf. – Novosibirsk: ICMMG, 1998. – P. 182–194.

[15] Piskunov S.V. WinALT – a simulation system for computations with spatial parallelism // NCC Bulletin, Series Comp. Comp. Science. – Novosibirsk: NCC Publisher, 1997. – Iss. 6. – P. 71–85.

[16] Denisov V.M., Matveev U.N., Ochin E.F. The principles of system organization for image processing on the basis of cellular logic // Foreign radioelectronics. – 1984. – № 1. – P. 3–25.

[17] Kostsov E.G., Piskunov S.V. Computer-aided design of two-layer computational matrix with optical interconnections // Avtometriya. – 2000. – Vol. 3. – P. 3–16.

[18] Pudov S. A method for learning of first-order cellular networks // NCC Bulletin, Series Comp. Comp. Science. – Novosibirsk: NCC Publisher, 2001. – Iss. 14. – P. 65–77.

[19] Ostapkevich M., Piskunov S.V. Basic constructions of models in WinALT // NCC Bulletin, Series Comp. Comp. Science. – Novosibirsk: NCC Publisher, 2001. – Iss. 14. – P. 43–58.

[20] Toffoli T., Margolus N. Cellular Automata Mechanics. – Massachusetts Institute of Technology, 1987; Russian Translation. – Moscow: Mir, 1991.

[21] Medvedev Yu. Cellular-automata models of fluid dynamics // NCC Bulletin. Special issue. – Novosibirsk: NCC Publisher, 1999. – P. 97–102.

[22] Ostapkevich M. Event-driven tools for open system design // NCC Bulletin. Special issue. – Novosibirsk: NCC Publisher, 1999. – P. 15–22.

[23] Ostapkevich M. Expulsive tree data structures for fast data search by a key // NCC Bulletin, Series Comp. Comp. Science. – Novosibirsk: NCC Publisher, 1999. – Iss. 10. – P. 73–82.

[24] Geist A., Beguelin A., Dongarra J., Jiang W., Manchek R., Sunderum V. PVM: Parallel Virtual Machine a User's Guide and Tutorial for Networked Parallel Computing. – Cambridge, Massachusetts, London, England: The MIT Press, 1994.

[25] Snir M., Otto S.M., Huss-Lederman S., Walker D., Dongarra J. MPI: the Complete Reference. – Boston: MIT Press, 1996.

[26] Network Interface Guide. – Sun Microsystems, Inc, Palo Alto, 2000.

[27] Technology of Electronic Communications: Multinetworks, Internetwork Communications, TCP/IP protocol. – Moscow: Ecotrends, 1993. – Vol. 3 (in Russian).

[28] Wilkinson B. Computer Architecture: Design and Performance. – London: Prentice Hall, 1991.