Automated Program Design – an Example Solving a Weather Forecasting Problem

Anatoliy Doroshenko, Pavlo Ivanenko, Olga Ovdii, and Olena Yatsenko*

Abstract: High-level algebra-algorithmic software tools for automated design of parallel code in the OpenMP environment are developed for the purpose of both producing efficient parallel code and increasing the performance of program developers. Application of the tools is illustrated with an example of a problem in atmosphere circulation modeling, represented as a service belonging to an Internet portal providing meteorological forecasting services. Results of execution of the parallel weather forecasting program on multiprocessor platforms are given.

Keywords: atmosphere circulation modeling; automated program design; multiprocessor platform; parallel computing; weather forecasting

PACS: 02.70.Wz, 02.70.Bf, 07.05.Tp

1 Introduction

One of the prospective directions in the development of parallel and distributed computing systems is the construction of program abstractions using high-level algorithmic languages and models. In previous works [1–5], we have been developing a theory, methodology and software tools for automated parallel program design, based on the algebra of algorithms. The algebra is applied for formalization, substantiation of correctness and transformation of algorithms. The developed algebra-algorithmic tools include the Integrated toolkit for Design and Synthesis of programs (IDS) [1, 3, 4] and its alternative version, the Synthesis framework [5], which is based on usage of Web technologies. The IDS toolkit is intended for automated design and generation of serial and parallel programs on the basis of high-level specifications of algorithms (schemes).

One of the main application domains of our tools is in weather forecasting. Forecasting systems are getting more complex every year as underlying mathematical models evolve and the amount of computed data grows swiftly. The problem of meteorological forecasting [6] is not an exception. Taking into account a large demand for meteorological prognoses in different fields of human activity, design and development of programs for solving this problem are extraordinarily timely. In addition, reliability and efficiency of meteorological forecasts are of high and sometimes critical importance, which puts high demands on the accuracy of the results and response time of forecasting programs. A large amount of computation over huge data arrays demands the usage of parallel computing.

In the previous paper [5], we considered an automated design of a parallel program for solving a simplified two-dimensional convection-diffusion problem, which arises in mathematical modeling of atmosphere circulation in meteorology [7]. The main goal of this paper is to develop algebra-algorithmic software tools for an automated design of efficient parallel code in the OpenMP environment. The application of the tools is illustrated by designing a program for the three-dimensional convection-diffusion problem. The developed parallel program is used in a Web portal for weather forecasting purposes that is under development [8]. In this work we also consider the automated designing of a service, which belongs to the portal and is implemented in the Java language.

The proposed approach is related to works on algebraic programming [9], synthesis of programs from specifications [10–12], automated generation of OpenMP programs [13–16] and Java applications [17–19]. In particular, works [13, 14] consider the tools for automatic OpenMP code generation, which is implemented using a polyhedral model, an algebraic abstraction which is used to detect data dependencies and perform high-level transformations of loops and other control flow structures for the...
purpose of program parallelization. Paper [15] proposes a methodology for translating a program written in Signal, which is a synchronous language dedicated for the functional description of safety critical systems, to OpenMP code. In [16] a source-to-source compilation framework for analyzing and transforming programs is considered, which is applied for automatic parallelization of scientific and signal processing applications. Paper [17] presents an approach to an automatic construction of Java programs from functional specifications, which is based on an inductive theorem prover. Works [18, 19] are devoted to Java code generation from a model-based design.

The main difference of our approach from the mentioned related works is that it uses high-level algebraic specifications which can be represented in three equivalent forms: algebraic, natural linguistic and graphical, and therefore giving a comprehensive understanding of specifications and facilitating achievement of demanded program quality. Another advantage of our tools is the method of automated design of syntactically correct algorithm specifications [1, 3, 4], which eliminates syntax errors during construction of algorithm schemes.

2 Automated Design of Programs using Algorithmic Algebra Facilities

Our approach to the design of algorithms and programs is based on the Glushkov System of Algorithmic Algebra (SAA) [1, 2], which is applied for formalized representation of algorithmic knowledge. SAA is the two-based algebra $\langle Pr, Op; \Omega \rangle$, where $Pr$ is a set of logical conditions (predicates) and $Op$ is a set of operators, both defined on an informational set; $\Omega$ is a signature of operations, which will be considered below.

The algorithms specified in SAA are called SAA schemes. The schemes can be represented in algebraic, natural linguistic (textual) and graphical forms [1].

2.1 The Main Operations of Algorithmic Algebra

Below we give the list of the main SAA operations, specifications of which are written in a natural linguistic form.

1. The operation of serial execution of two operators:
   
   "operator 1"
   
   "operator 2"

2. The conditional execution of operators (branching):
   
   IF ‘condition’ THEN
   
   "operator 1"
   
   ELSE "operator 2" END IF

3. The loop operation:
   
   WHILE ‘condition’
   
   LOOP "operator"
   
   END OF LOOP

4. The switch operation, which executes one of $n$ operators, if the value of the corresponding condition is true, and then breaks execution without checking the values of other conditions:
   
   SELECT
   
   ( ['condition 1'] → “operator 1”,
   
   ['condition 2'] → “operator 2”,
   
   ...
   
   ['condition n'] → “operator n” )

5. The operation of asynchronous execution of $n$ operators (threads), where $i$ is the index of the current thread:
   
   PARALLEL ($i = 0, \ldots, n - 1$)
   
   ( “operator i” )

6. The synchronizer operation, which delays the computation until the value of the specified condition is true:
   
   WAIT ‘condition’

The usage of the above operations will be illustrated in Section 3.

2.2 The Operations of Algorithmic Algebra for the Object-Oriented Paradigm

SAA also contains operations for the formalization of the main concepts of object-oriented programming, such as classes, objects, etc. For compatibility with the Java programming language, the operations for the definition of annotated classes, fields, methods and parameters were
included in the signature. An annotation is a special form of syntactic metadata, which can be added to a source code and used for analysis, compilation or execution of code [20]. The list of the SAA operations for the object-oriented paradigm, used in this work, is given below.

1. The definition of an annotated class (in generalized form):

   Annotated class X implements Y (Class name, Interface name)

   <Annotations>
   Annotation (Annotation text 1);
   Annotation (Annotation text 2);
   ...
   <Fields>
   operators
   <Methods>
   operators

   Here Class name is a class name, which implements an interface with identifier Interface name; <Annotations>, <Fields>, <Methods> are the strings which precede the operators for definition of one or more annotations, data fields and methods of the class, respectively; Annotation text is a text of an annotation.

2. The annotations for classes, data fields and methods are defined with the help of the operator Annotation (Annotation text),

   where Annotation text is the parameter, in which the text of the annotation is specified.

   An example of the definition of an annotation for a class is Annotation (Service).

3. The definition of an initialized data field of a class:

   Field initialized (Modifiers, Field type, Field name)

   <Initialization>
   operator

   This construction is used for specifying a class field, which is assigned some initial value. Here Modifiers is a list of access modifiers (e.g. public, static, final); Field type is a type of the data field; Field name is the identifier of the field; <Initialization> is the string preceding the operator or an expression, the value of which is used for initializing the field.

4. The construction for definition of an annotated data field of a class is the following:

   Field annotated (Modifiers, Field type, Field name)

   <Annotations>
   Annotation (Annotation text 1);
   Annotation (Annotation text 2);
   ...

   Here the meanings of the parameters Modifiers, Field type and Field name are the same as in the operation given above; <Annotations> is a string, after which the operators for the definition of one or more annotations are to be given.

5. The definition of an annotated method of a class:

   Method annotated (Return type, Method name, Parameters list),

   <Annotations>
   Annotation (Annotation text 1);
   Annotation (Annotation text 2);
   ...
   <Body>
   operators

   Here Return type is the data type of the value returned from a method; Method name is the identifier of a method; Parameters list is a list of formal parameters; <Annotations> is a string, which precedes the operators for definition of annotations (e.g. Override, Transactional); <Body> is a string, which precedes the operators of a method body.

   A formal parameter of a method, which is specified in the Parameters list, is defined in the following way:

   ParamType ParamName,

   where ParamType is a data type of a parameter (variable); ParamName is the identifier of the parameter. The data types include basic types (short, int, long, float, double, char, String, boolean) and compound types (class objects and arrays).

   Non-annotated classes, data fields and methods are defined with the help of constructs, similar to given above.

6. The construction for calling a method of a class instance (an object):

   Call instance method (Instance name, Method name, Arguments),

   where Instance name is an identifier of a class instance; Method name is the identifier of the class method; Arguments is a list of parameters of the method.

   The usage of the operations considered in this subsection will be illustrated in Section 4.
2.3 The Tools for Automated Software Design

Our approach to an automated design of programs is based on a method which ensures the syntactical regularity of algorithm schemes [1, 3, 4]. The method consists of unfolding the design of schemes by superposition of predefined SAA language constructs (such as presented above in previous subsections), which are considered as reusable components and are stored in the database. The design process is represented by a tree of an algorithm. The user selects SAA constructs from the list and adds them to an algorithm tree (for example, by using a drag-and-drop technology). On each step of the design process, the system allows a user to select only those operations, the insertion of which into a scheme does not break its syntactical correctness. An algorithm tree is then used for automatic generation of SAA scheme text and a source code in a target programming language. The mapping of SAA operators to a text in a programming language is defined in the form of code templates in the database.

We implemented the aforementioned approach in the IDS toolkit [1, 3, 4] and its alternative version, called the Synthesis framework [5]. The IDS toolkit is intended for automated design of algorithm schemes and generation of programs in target programming languages (Java, C++). IDS integrates three forms of design-time representation of algorithms: algebraic, textual and flowchart. The main difference between the Synthesis framework [5] and IDS is that Synthesis is based on using Web technologies and is suited for object-oriented programming, whereas the IDS toolkit is mainly aimed at an imperative paradigm.

The IDS system consists of the following main components (see Figure 1):

- the constructor, which is applied for an automated design of algorithm schemes and generation of programs;
- the flowchart editor [1], intended for editing the graphical representation of an algorithm scheme;
- the parameter-driven generator of algorithms on the basis of higher level schemes, called hyperschemes [3];
- the database, containing the description of SAA operations, basic operators and predicates, and also their program implementations.

To automate the transformation (e.g. parallelization) of algorithms and programs, the IDS system is used together with the term rewriting system Termware [4, 21].

In previous works [1–5], the SAA structure and developed software tools were applied in automated design and generation of parallel programs for multicore central processing units and graphics processor units, in particular, in the subject domain of weather forecasting.

In the next sections, we further develop these in the direction of their application for the three-dimensional case of the meteorological forecasting problem, and also for automation of software development for an Internet portal providing weather prediction services.

3 Design of a Parallel Algorithm for Solving a Convection-Diffusion Problem

In this section we illustrate the usage of SAA and the IDS toolkit [1, 3, 4] on an example of developing a parallel program for a numerical solution of a convection-diffusion problem (CDP).

3.1 The Mathematical Model of the Convection-Diffusion Problem

The mathematical model of CDP [22] is a combination of the diffusion and convection equations and describes physical phenomena where particles, energy, or other physical quantities are transferred inside a physical system due to two processes: diffusion and convection.

Here we will consider the three-dimensional CDP, which arises in mathematical modeling of atmosphere circulation in meteorology [7]. The problem to be solved is presented as the set of the following convection-diffusion equations:

\[
\begin{align*}
\frac{\partial u}{\partial t} + \frac{u}{a \cos \phi} \frac{\partial u}{\partial \lambda} + \frac{v}{a} \frac{\partial v}{\partial \phi} + w \frac{\partial v}{\partial z} &= -\frac{1}{\rho a \cos \phi} \frac{\partial p}{\partial \lambda} \\
&+ (v \sin \phi - w \cos \phi) \left(2 \omega + \frac{u}{a \cos \phi}\right) + \frac{1}{\rho} F_{\lambda},
\end{align*}
\]  

(1)
The parallel numerical implementation of the above model (1)–(5) is based on the following algorithm which parallelizes the task over three levels:

- the equations (each equation is solved independently);
- space directions \( \lambda, \phi \) and \( z \);
- the subdomains for each space direction. The computation domain is decomposed along \( \lambda \) for the sub-tasks in the directions \( \phi \) and \( z \) and along \( \phi \) for sub-tasks in the direction \( \lambda \).

The algorithm uses the modified additive-averaged splitting method [23]. The general scheme of the calculation is iterative. The input data determine the state of an atmosphere at the initial moment of time \( t_0 \). To obtain the prognosis for some moment \( \text{CALC}_{\_}\text{TIME} > t_0 \) the time segment is divided into \( \text{TmKnotsPer12h} \) relatively short segments of size \( \tau = \text{CALC}_{\_}\text{TIME}/\text{TmKnotsPer12h} \). At each iteration, the algorithm calculates the state of the atmosphere after the time \( \tau \) and its output data become input (initial) data for the next iteration. Increase of the step \( \tau \) results in a reduction of the number of iterations but leads to an increase in prognosis error, which is why it is recommended that the value of \( \tau \) be chosen from the interval \([0.5, 10]\) seconds. At each iteration, the equations for each coordinate direction are solved independently and then the results are gathered and processed, which requires synchronization. The usage of the modified additive-averaged splitting method allows the combination of several iterations (let’s denote the number of iterations by \( m \)) into one, which reduces the synchronization overhead. Increase in value of the parameter \( m \) also results in loss of adequacy of the solution and the recommended interval of values is \( m \in [1, 10] \).

Below, we give the generalized SAA scheme for the parallel algorithm developed for the solution of the convection-diffusion problem. The scheme was designed in the IDS toolkit with the usage of SAA language constructs which were described in Section 2.1.

**SCHEME “Parallel Algorithm for Solving the 3-Dimensional Convection-Diffusion Problem”**

“Main scheme” =

“Set initial parameters (Subdomains, M\_prm, TmKnotsPer12h, TmLimCalc, tau);”

“Start the timer”;

“Load the data to arrays for \((p, u, v, w, T)\);”

\((\text{Proc\_Num} := \text{“Parallel computation”});\)

“Stop the timer and print the execution time”;

“Compare the computed values with actual values and calculate an error”;

“Save the data to files for the 3-dimensional convection-diffusion problem”;
F, C and the boundary conditions”; SELECT
( ['(dir) = (0)'] →
  “Compute the subtasks for (Lam) direction”,
)
['(dir) = (1)'] →
  “Compute the subtasks for (Fi) direction”,
['(dir) = (2)'] →
  “Compute the subtasks for (Z) direction”
);
  “Interchange values of intermediate arrays”;
  WAIT ‘All threads completed work’;
  “Compute the average on the basis of the results obtained for each direction”;
  WAIT ‘All threads completed work’;
  “Storing the results to global arrays”;
  WAIT ‘All threads completed work’;
  “Increase (n) by (M prm)”;
END OF LOOP;
  “Deallocation of memory for arrays for 3-dimensional convection-diffusion problem”
);
  “Return value (NmbThreads)”;

END OF SCHEME

The above scheme consists of two parts, “Main scheme” and “Parallel computation”, which are compound operators and correspond to subroutines in a program to be generated in a target programming language (C++).

The compound operator “Main scheme” is composed of the operators: setting the initial parameters of the algorithm; measuring the execution time of the program; loading the data into arrays in which the main values of the problem are stored; calling the “Parallel computation” compound operator; comparing the computed values and saving the results data to files. The main scheme contains the following variables: Subdomains ≥ 1 is the number of subregions for each of the space directions; M prm ∈ [1, 10] is the parameter (m) of the modified additive-averaged splitting method [23]; TmKnotsPer12h is the number of points in the time mesh for a twelve-hour period; TmLimCalc is the final value of time (in seconds) in the condition of the while loop; tau is the time step (in seconds), which is defined according to the formula: tau = CALC_TIME/TmKnotsPer12h, where CALC_TIME is the end time for calculating the prognosis; u, v, w are the components of the wind velocity vector; p is atmospheric pressure; T is absolute temperature; Proc Num is the number of threads.

The compound operator “Parallel computation” is composed of the initialization operators and the parallel region. The parallel region contains the operators of initialization of variables and arrays and the m-loop of the modified additive-averaged splitting method [23]. The beginning of the m-loop body contains the preparation for the main computation, namely, setting current values for u, v, w, p and boundary conditions, calculation of equation coefficients, etc. Then the subtasks for each direction λ, φ and z are computed. Next, the average values are calculated based on the results obtained for each direction. At the end of the loop, the results are saved to global arrays.

The meanings of the main constants and variables of the parallel scheme are as follows. The variable NmbThreads is the number of threads, which is calculated according to the formula:

\[
NmbThreads = \text{Subdomains} \times \text{DIR} \times \text{EVL\_MT\_VAL}
\]  

where DIR = 3 is the number of space directions (λ, φ, z); EVL\_MT\_VAL is the number of meteorological values in the evolution equation.

The variable proc in the scheme is the index of the current thread; n is a loop variable; D is air density; F is an equation coefficient; C is an advection coefficient; dir is an integer variable for storing the value of the space direction being processed (0 for λ, 1 for φ and 2 for z); Lam, Fi and Z are the parameters, which correspond to space directions.

The IDS toolkit automatically translated the above SAA scheme into C++ source code with the usage of OpenMP directives [24]. The translation process is based on usage of program code templates which are defined for each high-level SAA construction and stored in the database of IDS. In particular, the SAA operation of asynchronous execution of operators (PARALLEL) was implemented using the OpenMP construction “#pragma omp parallel”. The fragment of C++ source code, generated for this operation, is given below.

```c++
// PARALLEL (proc = 0, ..., NmbThreads−1)
omp_set_num_threads(NmbThreads);
#pragma omp parallel
{
  // Get thread index:
  int proc = omp_get_thread_num();
```
// Get the total number of threads:
#pragma omp master
{
    NmbThreads = omp_get_num_threads();
    printf("\n Number of threads: \%d. \n", NmbThreads);
}
...

The synchronizer operation (WAIT) is translated into the "#pragma omp barrier" directive.

4 Designing the Online Weather Forecasting Service

In this section, we illustrate the usage of SAA and the Synthesis framework [5] on the example of designing a fragment of an online service, which is part of the system developed here for providing meteorological forecasting services [8]. The system consists of the components shown in Figure 2.

The Web Portal provides the user interface for accessing the services of the system. It was implemented using a free open source enterprise portal software product Liferay [25].

The Service for Managing Meteorological Prognoses is a service which performs the functions of planning, synchronizing and managing the system. It receives and processes the requests which contain the information about time, place, and type of prognosis.

The Providers of Meteorological Data are one or several providers of the initial meteorological data which are the basis for computing prognoses. At present, the initial data are provided by the meteorological center located in Offenbach, Germany [26].

The Providers of Prognoses are external systems which provide meteorological forecasting services. The present implementation of our system uses the OpenWeatherMap service [27] as an external prognosis provider.

The Database of the system contains the initial meteorological data and the resulting data of prognoses which are computed on the basis of initial data. PostgreSQL [28] was used as the database management system.

The Service for Computing Meteorological Prognoses performs the calculation of prognoses with the usage of the parallel program which is executed on a multiprocessor platform (a cluster).

Most of the system was written in Java, except for parallel programs which implement numerical weather prediction which were written in the C++ language with the usage of OpenMP. The programs implement the solution of the two-dimensional and three-dimensional convection-diffusion problem. The design of the algorithm scheme for the three-dimensional case of the problem was considered above in Section 3.

Below we give the SAA scheme of the Java class Prediction3DServiceImpl, which is one of the software components of the weather forecasting system considered here. The methods of this class are intended for finding, saving, creating and deleting prognoses, which are stored in the database of the system. The scheme was designed in the Synthesis framework [5] with the usage of the SAA language constructs described in Section 2.2.

Annotated class X implements Y (Prediction3DServiceImpl, Prediction3DService)
<Annotations>
  Annotation (Service)
<Fields>
Field initialized (private static final, Logger, LOGGER)
  <Initialization>
    Call instance method (Logger, getLoader, Prediction3DServiceImpl.class);
Field annotated (public, Prediction3DRepository, prediction3dRepository)
  <Annotations>
    Annotation (Resource)
<Methods>

Method annotated (Prediction3D, findById, Long id)

<Annotations>
Annotation (Override);
Annotation (Transactional)

<Body>
Call instance method (LOGGER, debug, “Trying get prediction3d #" + id + “from DB”);
Return (Call instance method (prediction3dRepository, findById, id));

Method annotated (void, savePrediction, Prediction3D prediction)

<Annotations>
Annotation (Override);
Annotation (Transactional)

<Body>
Assign (prediction, Call instance method (prediction3dRepository, save, prediction));
Call instance method (LOGGER, debug, “Prediction3D #" + prediction.getId() + “has been saved to DB”);

Method annotated (Long, createPrediction, Prediction3D prediction)

<Annotations>
Annotation (Override)

<Body>
Assign (prediction, Call instance method (prediction3dRepository, save, prediction));
Return (Call instance method (prediction, getId));

Method annotated (void, deletePrediction, Long id)

<Annotations>
Annotation (Override)

<Body>
Call instance method (prediction3dRepository, delete, id);

In the above scheme, the class Prediction3DServiceImpl is defined using the construction “Annotated class X implements Y” (see Subsection 2.2). The parameters of the construction are the class name and the interface name (Prediction3DService). After the class header, the class annotation and the definitions of class fields and methods are given.

The class fields are the initialized field LOGGER and the annotated field prediction3DRepository. The LOGGER field is an instance of the Logger class, the methods of which are intended for writing the messages to a journal (log). The prediction3DRepository field presents the database of the system.

The Prediction3DServiceImpl class contains four methods. The findById method is aimed at searching the computed prognosis in the database according to id number. The methods savePrediction, createPrediction and deletePrediction are intended for saving, creating and deleting prognoses from the database, respectively.

The Synthesis framework automatically translated the above scheme into the following Java source code of the Prediction3DServiceImpl class:

```java
@Service
public class Prediction3DServiceImpl implements Prediction3DService {

    private static final Logger LOGGER = Logger.getLogger(Prediction3DServiceImpl.class);

    @Resource
    public Prediction3DRepository prediction3dRepository;

    @Override
    @Transactional
    public Prediction3D findById(Long id) {
        LOGGER.debug("Trying get prediction3d "+ id + "from DB");
        return prediction3dRepository.findById(id);
    }

    @Override
    @Transactional
    public void savePrediction(Prediction3D prediction) {
        prediction = prediction3dRepository.save(prediction);
        LOGGER.debug("Prediction3D "+ prediction.getId() + "has been saved to DB");
    }

    @Override
    @Transactional
    public Long createPrediction(Prediction3D prediction) {
        Unauthenticated
        Download Date | 6/1/19 10:43 PM
```
5 Experiment Results

We carried out two independent experiments, which consisted of execution of the OpenMP program developed here for solving the weather forecasting problem, on the following two multiprocessor platforms:

- a node of the cluster of the Institute of Software Systems (ISS), which contains two Quad-Core Intel Xeon E5405 processors (8 cores in total on the node);
- a node of the SCIT-4 supercomputer of the Institute of Cybernetics [29], which contains two Intel Xeon X5675 processors (altogether 12 physical cores, or 24 logical cores in hyperthreading mode, on the node).

During the experiments we used the following values of program parameters (see Section 3): the time for calculating the prognosis $\text{CALC\_TIME} = 43200$ seconds (12 hours); the parameter of the modified additive-averaged splitting method $\text{M\_prm} = 10$; the number of time knots $\text{TmKnotsPer12h} = 4320$; the final time $\text{TmLimCalc} = 43200$ seconds (12 hours); the time step $\text{tau} = 10$ seconds. The space domain of the problem definition was $\lambda \in [0^\circ, 90^\circ], \phi \in [0^\circ, 90^\circ], z \in [8000; 18000]$. The size of the input finite-difference mesh was $182 \times 182 \times 11$ points. The number of threads was set according to the formula (6), given in Section 3.2: $\text{NmbThreads} = \text{Subdomains} \ast 3 \ast 3$.

Figure 3 shows the obtained values for the efficiency of the parallel program

$$E = \frac{Sp}{N},$$

where $Sp = \frac{T(1)}{T(M)}$ is the multiprocessor speedup, $T(1)$ denotes the execution time of the corresponding sequential program, $T(N)$ is the execution time of the parallel program, $N$ is the number of cores. On ISS cluster the number of cores $N = 8$ and on SCIT-4 $N = 12$.

![Figure 3: The values of the efficiency of the parallel program obtained on SCIT-4 and ISS clusters at three values of the number of subdomains.](image)

The program was executed at three values of $\text{Subdomains}$: 1, 2 and 3; the number of threads $\text{NmbThreads}$ was set to 9, 18 and 27, correspondingly.

On SCIT-4, the maximum efficiency 0.98 was obtained at the value of $\text{Subdomains} = 2$, which is optimal at the currently used number of processor cores. On ISS, the maximum efficiency was 0.74 at $\text{Subdomains} = 3$. The higher efficiency obtained on the SCIT-4 cluster is explained by the presence of hyperthreading.

6 Conclusion

We developed software tools for an automated design of efficient parallel code in the OpenMP environment. The distinctive feature of our approach consists of using high-level algebra-algorithmic program specifications which are represented in a natural linguistic form. The developed tools automatically translate the specifications into source code in a programming language. The usage of the proposed toolset is illustrated with an example of parallel program development for solving a problem of atmosphere circulation modeling. The program is used in an Internet portal which provides meteorological forecasting services.

References


[23] Prusov V.A., Doroshenko A.E., Chernysh R.I., Choosing the parameter of a modified additive-averaged splitting algorithm, Cybernetics and Systems Analysis, 2009, 45, 589-596


