Concept of Distributed Processing System of Images Flow in Terms of π-calculus

Aleksey Kondratyev, Igor Tishchenko
Program Systems Institute of Russian Academy of Sciences
Pereslavl-Zalessky, Russian Federation
ronkajitsu, igor.p.tishchenko@gmail.com

Abstract—**The paper describes a concept of software tools for data stream processing. The tools can be used to implement parallel processing systems. Description of the task is presented in the first part of paper. The system is based on pipeline parallelism and was distributed for using on a cluster computer. The paper describes a base scheme and a main work algorithm of the system. Tasks are presented in π-calculus terms. The system process tasks as π-calculus automata. An actual application example is presented.**

I. Introduction

Active development of parallel computing is accompanied by the growth of computing power and the creation of high-performance systems. Creation of supercomputers has several objectives:

- The maximum capacity (Top 500).
- The solution to a specific problem (data storage, data processing).

Supercomputers are not used for playing games or watching movies, so the majority of computer users are not interested in them. No one would create a cluster computer without a goal because of the high production and maintenance costs. Clusters without load usually are switched off completely or partially. We need high-quality custom software that is designed to perform a specific task which will make a full use of a cluster computer. In most cases users create this software "from scratch" by using libraries for parallel programming (MPI, Ocaam, OpenMP, PVM, etc.). Highly skilled professionals who can produce the correct decomposition of tasks for parallel execution and its correct implementation are required in order to develop effective programs to fully utilize available hardware.

One of many problems which require capabilities of supercomputers is the problem of data flow processing. The data may be from sensor nodes with wide spectrum. The data processing could consist of several stages like searching, tracking or monitoring object state, etc. The main condition is a possibility to schematize task as a block-scheme of processing stages.

II. General system information

To split the computations into independent parts it's essential to analyze the task solution scheme. Equal computation volume and minimization of dependencies are the main requirements for splitting. In most cases analysis and task splitting are complex problems. To use the low-level libraries in a new supercomputer application it's important to have highly qualified specialists and a lot of time. The best way is to use special software tools in most cases [1].

Our program system has been successfully used in various scientific fields, and allows quickly creating applications oriented towards running on cluster nodes. Some of them could be named as an example:

- processing of remote sensing data [2],
- processing of medical information [3],
- designing and using of neural networks [4], [5],

The system allows dividing roles between users and developers. Users can solve a task without writing program code in contradistinction to typical ways. User may not know how to split algorithm into parts and make parallel program. They could just build a linear scheme of data processing which consists of program modules with data channels and initial conditions (see Fig. 1). Users can describe the whole data process algorithm by modules. The scheme will have modules with typical connection slots. It may be done by the graphical user interface (GUI) (see Fig. 2). GUI has all the information about modules, like a list of available modules and channels. It allows users to make some part of work using their intuition and detect some errors on the design stage.

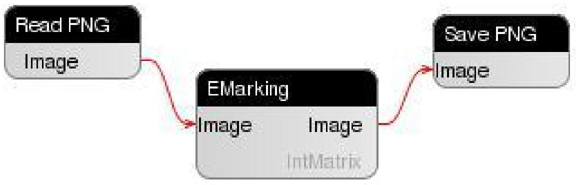

Fig. 1. Scheme of image regions isolation

The system algorithms could help reach calculation acceleration on supercomputers. As an example lets consider a problem of repainting a map based on texture classification. Results of the tests are shown in Table II and are illustrated on Fig. 3 and Fig. 4.

The textural classifier provides results inaccessible for point classifiers. But it requires more hardware resources. The system parallelism could provide close to linear acceleration as shown on Fig. 3. As shown on the Fig. 4, processing time can be easily decreased in some cases.

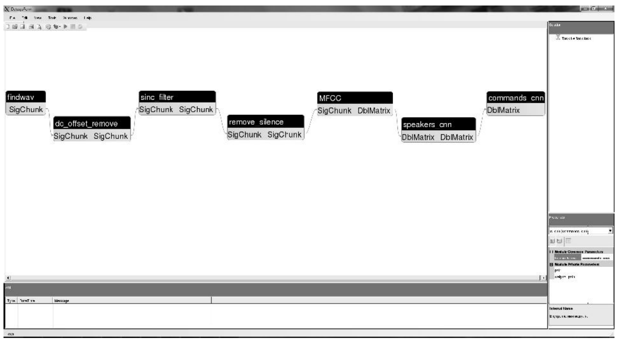

Fig. 2. Graphical user interface

TABLE I. THE PARALLEL PROGRAM EFFECTIVENESS

CPU kernel count	Time	Acceleration
1	185.7	1
2	93.9	1.98
3	64.8	2.86
4	51	3.64
5	47.7	3.89
6	33	5.48

The field of parallel processing is very well studied. A lot of software tools were created for a lot of applications. Some of them are completed software. Some of the software tools can be used for implementation of parallel data processing systems: CODE, HeNCE, GRADE, TRAPPER, Kepler, etc. But all of them require knowledge about parallel programming. Most of them are only GUI for parallel programming library. And only Kepler allows implementing applications with conveyor. Other software does not allow this. The described tools simplify the development of new data processing systems rather than trying to achieve maximum performance.

III. KEY FEATURES

The system is based on a modular structure. Each component has its own role in the system but all of them are intended for one goal. The system is the environment for running program modules in the pipeline parallelism mode. Principal working scheme is presented on Fig. 5.

Each module is a minimal program with some processing function and may have several input and output channels. The system considers each module as one function with input parameters. It provides an ability to organize calculation process with pipeline parallelism.

In the traditional work-sharing approach, threads run different iterations concurrently [6], [7], [8], [9]. Each thread runs iteration in its entirety and then proceeds to the next iteration. An alternate approach is to use pipeline parallelism where we split loop iterations into stages and threads operate on different stages from different iterations concurrently (see Fig. 6).

Pipeline parallelism is powerful because it can expose parallelism in ordered loops where iterations are non-independent and cannot run concurrently. By splitting each loop iteration into segments, we can expose intraiteration parallelism. When multiple cores are assigned to a pipeline stage, its throughput increases (ideally) linearly. Thus, a common mechanism to speed up a pipeline workload is to assign more cores to

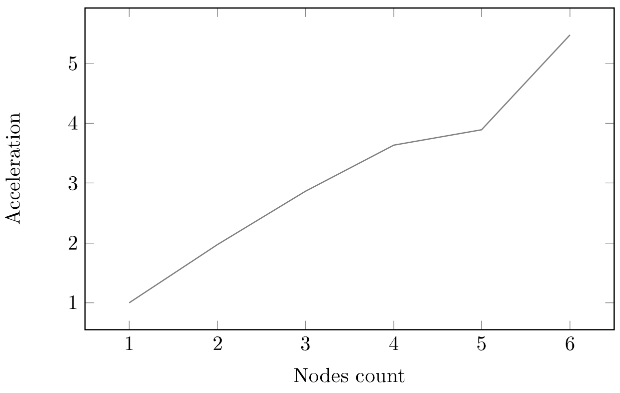

Fig. 3. Speedup of the program

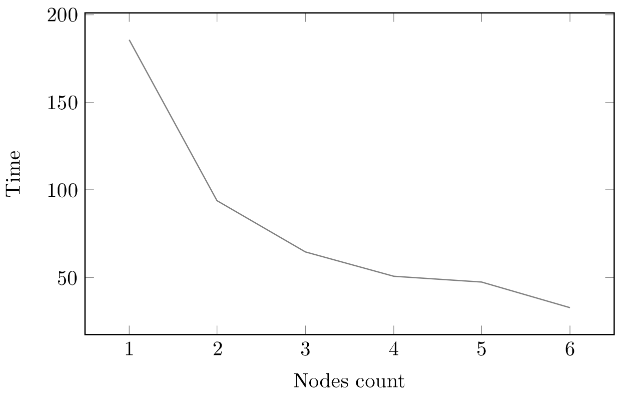

Fig. 4. The parallel program results

the slowest stages to balance the throughput. However, programmers are often unable to balance pipelines completely which leads to thread waiting, thus, lost performance opportunity [10].

This work is performed by the system. The system transforms a task into a list of system commands. The system can be described by a state machine. Each command is a conditional transition between its states. Each command describes some action like data transfer or module execution. System scheduler strives to use all available resources at any time [11]. With the release of hardware resources scheduler must choose a command which can be executed in that moment from the command queue. This method proved to be good for modules with considerable time of processing for little amount of data. So the time for command selection is smaller than processing time.

When we talk about image stream processing we have a lot of fast processing steps. If we try to test it as a part of the system we get a terrible result with multiple nodes. The time of processing and transfer data is smaller than command selection time. Similar type of problems must be solved by another method of scheduling and data management. The main idea of that method lies in phrase “transfer data when needed”. A system module is described as a “pure” function. It means that a module has two input parameters. One of them is data for processing and another one is an internal module state with module initial parameters. The system tries to process the data locally for cluster node. Schematically the basic actions could

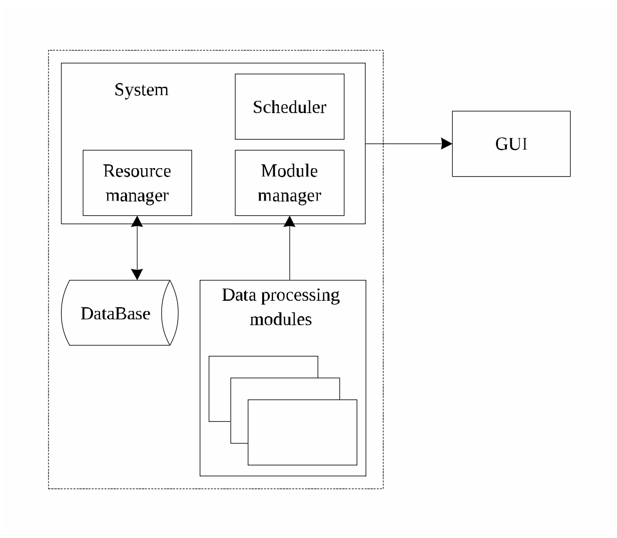

Fig. 5. Principal working scheme

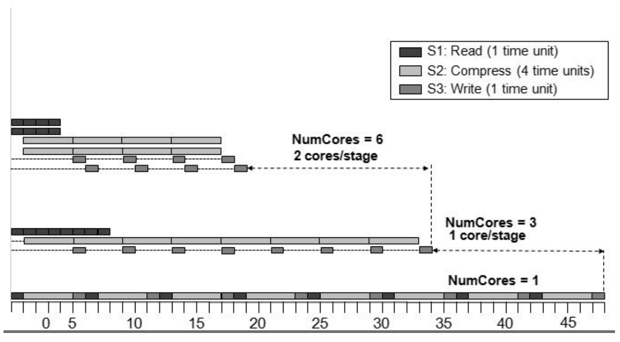

Fig. 6. Pipeline parallelism

be described in several steps:

1) System initialization.
2) Waiting for new task.
3) Generating commands from input task.
4) Module initialization on cluster nodes.
5) Command executing with rules:
    a) Processing data on one node with several modules.
    b) Spread data from stream on several nodes.
    c) Combine data transmissions.
    d) Minimization of data transmissions.
6) After the completion of the task go to step 2.

The importance of the data distribution for processing among cluster nodes is proved by various papers. Preliminary data distribution can significantly reduce the processing time [12], [13], [14].

IV. SYSTEM MODEL

The Worker concept is closely linked with the concept of a computing node. The Worker is the same thing as computing node where it was launched. It describes computing resources, data processing modules from those nodes. Also the Worker could execute commands from command list which describes data processing computation. Worker state consists of information about usage of CPU, memory, hard drive and data processing modules activity. The Scheduler decides commands for process by using information about resources from all known Workers. It generates commands for Workers from incoming data processing task description. Scheduler does task decomposition, load balancing and provides a mechanism for module communication in computing environment. It has information about computing environment — computing node list with information: CPU load, memory usage, network usage, connection and module state, threads status.

The Worker could execute most of data processing modules. It depends on the Worker capabilities. As example could be named GPU support.

The primary way to realize application is data processing module. The module is realization of data processing function or data generation function. The module is divided into several parts:

- internal state of the module;
- input and output data channels;
- module initialization function;
- data processing function.

A. Scheduler

Load balancing and task execution is based on system of control commands. The work of scheduler could be described in several steps:

1) insert incoming task to queue,
2) task decomposition into list of control commands,
3) insert commands to command queue,
4) search command list for execution in accordance with limitations (minimization of data transmissions, maximization of computing resource utilization, command priorities);
5) send command to execution on Workers,
6) wait for results from Workers,
7) return to step 4.

As result from Worker could be received an alert or new command for new data processing. That command must be added to command queue. Scheduler could be described as automata (Fig. 7) S = (Q, S, δ, Start, End) where:

- Q = {Start, Init, Update, Select, Apply, Wait, End} — finite number of states, which describes phases of scheduler,
- S = {CONTINUE, SELECT_CMD, SENT_CMD, NO_CMD, EVENT, EXIT_CMD}.
- $\delta : QxX \to Q$ — transition,
- Start — start state,
- $F = \{End\}$ — set of final states.

States of automata for scheduler are:

- Init — system initialization,

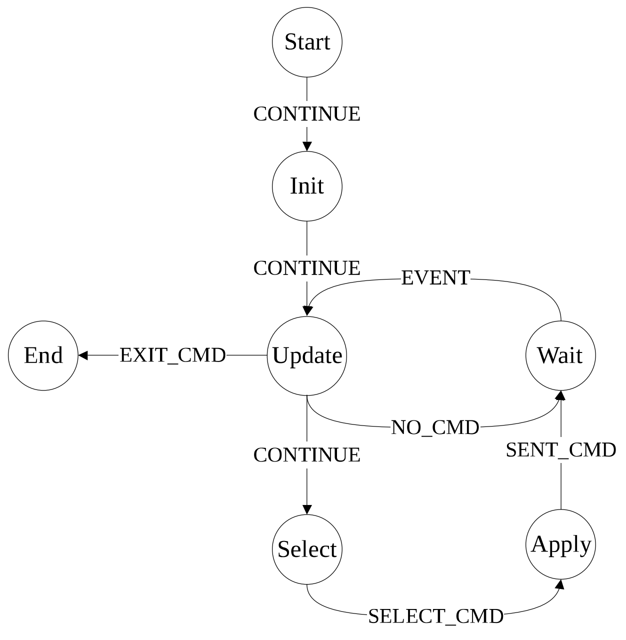

Fig. 7. Scheduler state automata

- Update — system information update (hardware resources, data packages, generation of new control commands),
- Select — selection of control command and Worker,
- Apply — applying of command (command transmission and execution),
- Wait — waiting for response from Worker.

B. Worker

Worker is a data processing node in terms of the data processing system. Worker has a set of data processing modules, pool of threads for data processing (one for each CPU thread). The main Worker goal is data processing module execution. The Worker executes control commands from Scheduler. The Worker could send alert to Scheduler as result of command execution. The command set for Worker is limited and strictly defined:

1) init worker command,
2) load module command,
3) init module command,
4) module execution command,
5) data transmission command,
6) stop module command,
7) delete data command,
8) stop worker command.

Each command calls Worker function with several actions:

- Init worker command is the first command for Worker from Scheduler. Worker creates pool of threads, loads information about dynamic libraries for resources and modules. Worker send message to Scheduler about actions with information about node (thread count, module list and etc.). After that Worker is ready for work.
- Load module command is the command for loading module with parameters: identification number, custom module name. Worker loads module and create a copy with parameters. Also Worker loads information about data channels. Worker send message about successfully done actions.
- Init module command is the command for module initialization with custom parameters from user task description. It calls function “init()” of data processing module. It allows module to do some actions one time before data processing (like GPU context initialization). Worker send message about successfully done actions.
- Module execution command transfer to Worker when it has capabilities for command execution (hardware resources, module description, data for processing). Worker calls module function “work()” for each data in incoming channels. Result of function execution may be new data for processing. Worker send message to Scheduler about actions result (such as id for new data). Function could be executed in 2 cases: module has data in input channel, module doesn’t have input channels.
- Data transmission command transfer to a Worker when it has data from one module for another data processing module. The Worker transfer data for processing from output channel to input channel throw local or network socket. After that Worker send message about successfully done actions.
- Stop module command transfer to Worker when some module must be forcibly stopped for some reasons. Worker stops module execution and delete all channels for this module.
- Delete data command is needed for removing data from Worker when this data was processed.
- Stop Worker command stops all active Worker data processing modules.

Worker could be described as finite-state machine (Fig. 8): S = (Q, S, δ, Start, End) where:

- Q = {InitW, Work, SearchThread, Wait, StopW} — finite number of states, which describes phases of Worker, S = {CONTINUE, RECEIVE_CMD, SEND_EVENT, EXIT_CMD}.
- $\delta : QxX \to Q$ — transition,
- InitW — start state,
- $F = \{StopW\}$ — set of final states.

States of automata for Worker are:

- SearchThread — select thread for command execution,
- Work — command execution: module loading and initialization, data processing and transferring,
- Wait — waiting of new command from Scheduler.

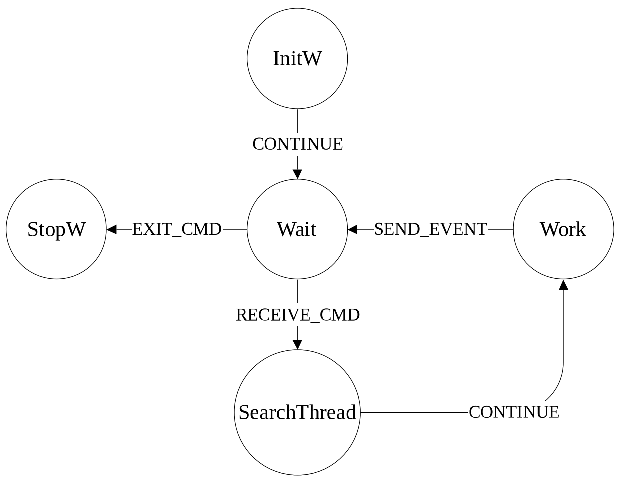

Fig. 8. Worker automata

V. Tasks

Data processing application is described by scheme of data processing. Scheme consists of data processing modules and transfer channels between them. Scheme describes data processing conveyor. Scheduler creates internal representation for task and verifies it. Task could be presented as directed acyclic graph (Fir. 9) where vertices are modules and edges are channels.

A. Data processing task

Usually task consists of several modules where could be operations branching operations and data combining. But task always has initial and finishing modules.

Scheme representation could be used for most cases. However for automatic verification and simulation we could use another way.

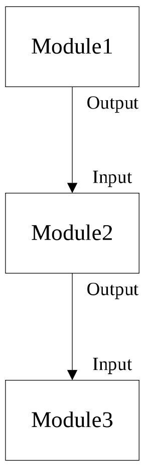

Fig. 9. Task graphical view

The scheme from fig. 9 we can see all kinds of modules:

- "Generator" — module without input channels. This type of modules should generate data in "work()" function. It may be image reader, camera capturer, database reader or etc. It "generate" data and send it to the output channel.
- "Processor" — module with input and output channels. This type of module receives data from input channel, processes it and sends some result to output channel.
- "Saver" — module without output channels. This type of module receives data from input channel and could somehow saves it (file, database) or just does some actions like camera rotation.

B. π-calculus

We will use pi-calclulus terms for task description. The π-calculus is a process calculus [15]. It allows channel names to be communicated along the channels themselves, and in this way it is able to describe concurrent computations whose network configuration may change during the computation. There are several reasons for this choice. The data processing module could be represented as single thread in the system where modules work in concurrency mode. In the case where we have unlimited count of computing nodes we can represent any data processing scheme to continuous conveyor without data buffering in channels. π-calculus is model of message passing. There is no any other way to communicate as global variable or etc. Transition from one state to another is associated with message sending. It corresponds to the behavior of the system. Any system action (command) is associated with data or message. Execution of "work()" function comes after data arrive in input channel and ends with data in output channel.

Central to the π-calculus is the notion of name. It's unlimited and doesn't have any structure. But names play as communication channels and variables. The process in π-calculus is following:

- $c(x).P$ — input prefix, receiving data "x" from channel "c",
- $\bar{c}\langle x\rangle.P$ — output prefix, sending data "x" to channel "c",
- $P|Q$ — two processes or threads executed concurrently,
- $!P$ — process replication,
- $(vx)P$ — creation of a new name "x",
- τ_p — internal process action,
- 0 — the nil process,
- $+$ — operation combining data.

C. Task in π-calculus

Scheme conversion is based on several constructions:

- Linear section (Fig. 10). It's used for communication of two modules and could be represented with next equations:

$$\begin{aligned} P &= A|B; \\ A &= \tau_A.\bar{b}\langle x\rangle.0; \\ B &= b(x).\tau_B.B'; \end{aligned} \tag{1}$$

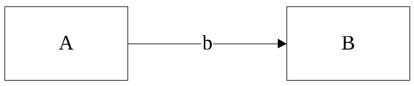

Fig. 10. Linear section

- Bifurcation section (Fig. 11). It could be represented with next equations:

$$\begin{gathered} P = A|(B|C); \\ A = \tau_A.(\bar{b}\langle x\rangle.0|\bar{c}\langle x\rangle.0); \\ B = b(x).\tau_B.B'; \\ C = c(x).\tau_C.C'; \end{gathered} \tag{2}$$

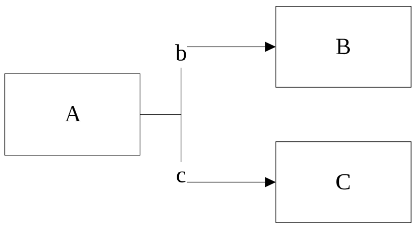

Fig. 11. Bifurcation section

- Combination section (Fig. 12). It could be represented with next equations:

$$\begin{gathered} P = (B|C)|D; \\ B = \tau_B.\bar{d_1}\langle x\rangle.0; \\ C = \tau_C.\bar{d_2}\langle x\rangle.0; \\ D = d_1(x).d_2(x).D'; \end{gathered} \tag{3}$$

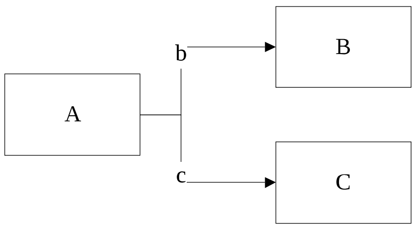

Fig. 12. Combination section

With described rules we can represent any scheme to formal view. As example we can see simple scheme (Fig. 13). Representation of scheme is due to the following rules and

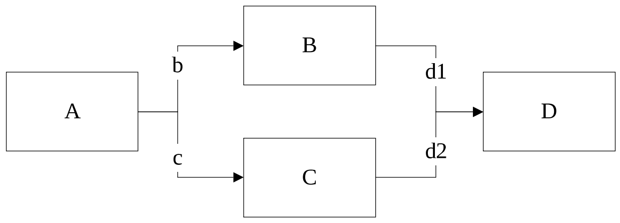

Fig. 13. Task scheme

steps:

- Modules must be named with letters.
- Channels also must be named with small letters similar to modules with next rules:
  - output channel must be named with same letter as module,
  - multiple output channels must have index after letter in their name.
- We must make representation of scheme with rules which are described before.

In this way the scheme is described with next equations:

$$\begin{gathered} P = !A|(B|C)|D; \\ A = \tau_A.(\bar{b}\langle x\rangle.0|\bar{c}\langle x\rangle.0); \\ B = b(x).\tau_B.\bar{d_1}\langle y\rangle.0; \\ C = c(x).\tau_C.\bar{d_2}\langle y\rangle.0; \\ D = d_1(y).d_2(y).\tau_D.0; \end{gathered} \tag{4}$$

Verification process consists of several steps. Step 1 is internal process in module "A" (data generation):

$$\begin{gathered} A = \bar{b}\langle x\rangle.0|\bar{c}\langle x\rangle.0; \\ B = b(x).\tau_B.\bar{d_1}\langle y\rangle.0; \\ C = c(x).\tau_C.\bar{d_2}\langle y\rangle.0; \\ D = d_1(y).d_2(y).\tau_D.0; \end{gathered} \tag{5}$$

Step 2 is data transfer from module "A" to modules "B" and "C" (process "A" is nil after that):

$$\begin{gathered} A = 0|0; \\ B = \tau_B.\bar{d_1}\langle y\rangle.0; \\ C = \tau_C.\bar{d_2}\langle y\rangle.0; \\ D = d_1(y).d_2(y).\tau_D.0; \end{gathered} \tag{6}$$

Step 3 is internal actions in modules "B" and "C":

$$\begin{gathered} A = 0|0; \\ B = \bar{d_1}\langle y\rangle.0; \\ C = \bar{d_2}\langle y\rangle.0; \\ D = d_1(y).d_2(y).\tau_D.0; \end{gathered} \tag{7}$$

Step 4 is data transfer from modules "B" and "C" to "D":

$$\begin{gathered} A = 0|0; \\ B = 0; \\ C = 0; \\ D = \tau_D.0; \end{gathered} \tag{8}$$

And the last step is internal actions in module "D":

$$\begin{gathered} A = 0; \\ B = 0; \\ C = 0; \\ D = 0; \end{gathered} \tag{9}$$

All processes are nil so it's the end. Task could be completed for data package.

The presented approach allows describing different schemes and tasks which includes linear sections, data bifurcation and combination. And the scheme (Fig. 13) consists of several steps with basic scheme constructions.

VI. Conclusion

The described system is useful software tools for users who don't have knowledge about parallel programming. The processing division into stages allow to process data independently with several threads. It may be perfect choice for quickly making data processing application for distributed system. Prepared scheduling algorithms allow obtaining acceleration in various problems.

The tools have several drawbacks. The best results can be achieved with coarsely granular concurrency. Scheduler has numerous leaks of time when choosing the next action. The other disadvantages are:

1) Not all algorithms are suitable.
2) Maximum load of computing resources is not always the best choice.

The concept has worked well for the computer network. The next stage is to use the concept in a heterogeneous computing environment. Various computing power of nodes requires a more sophisticated approach to load balancing.

π-calculus allows us to model and to show the system work in several threads. It shows data processing and work of conveyor. Also it allows us to verify application task scheme. Formalization of the task scheme allows creating various simulation models. It allows finding out eternal cycles in schemes and to guarantee the completion of calculations in finite time with modules which work finite time. With this concept the system is mechanism for concurrency run of multiple finite-states machines with π-calculus description.

Acknowledgments

This work was supported by the Ministry of Education and Science of the Russian Federation: #14.607.21.0088 agreement for a grant on "Development of methods and means of processing and intelligent analysis of images and dataflow obtained from a variety of stationary and mobile sensors, using high-performance distributed computing for the tasks of monitoring the premises and the surrounding area". Unique identifier: RFMEFI60714X0088.

References

[1] E. Rutledge, J. Kepner "PVL: An Object Oriented Software Library for Parallel Signal Processing", *in Proceedings of the 2001 IEEE International Conference on Cluster Computing (CLUSTER01)*, Web: http://www.computer.org/csdl/proceedings/cluster/2001/1116/00/11160074.pdf

[2] D.N. Stepanov, A.E. Kiryushina, E.S. Ivanov, A.A. Kondratiev "Software for pipeline parallel processing of remote sensing data in the cluster computing installations and graphics processing unit", *in Proceedings of Junior research and development conference of Ailama-zyan Pereslavl university*, Pereslavl, SIT-2014, 2014, pp. 5-20.

[3] V.F. Zadneprovsky, A.A. Talalaev, I.P. Tishchenko, V.P. Fralenko, V.M. Khachumov "Software tool complex high-performance image processing for medical and industrial use", *in Information technology and computer systems*, Vol. 1, 2014, pp. 61-72.

[4] A.A. Talalaev, I.P. Tishenko, V.P. Fralenko, V.M. Khachumov "Analysis of the efficiency of applying artificial neuron networks for solving recognition, compression, and prediction problems", *in Scientific and Technical Information Processing*, 2011, Vol. 38, pp. 313-321.

[5] A.A. Kondratyev "Parallel clustering of color images based on the self-organizing maps Ko-honen cluster using calculators", *in Proceedings of Junior research and development confe-rence of Ailamazyan Pereslavl university*, Pereslavl, SIT-2012, 2012, pp. 57-70.

[6] K. Czajkowski, I. Foster "A Resource Management Architecture for Metacomputing Sys-tem", *in Job Scheduling Strategies for Parallel Processing (JSSPP 1998): Proceedings of the 4th Workshop*, Orlando, Florida USA, March 30.

[7] F. Seredynski., A. Zomaya, "Sequential and Parallel Cellular Automata-Based Scheduling Algorithms", *in IEEE Transactions on Parallel & Distributed Systems*, vol. 13, Issue No.10, October, 2002.

[8] Kenli Li, Xiaoyong Tang, Keqin Li "Energy-Efficient Stochastic Task Scheduling on Hetero-geneous Computing Systems", *in IEEE Transactions on Parallel & Distributed Systems*, vol. 1, doi:10.1109/TPDS.2013.270.

[9] M. Pricopi, T. Mitra "Task Scheduling on Adaptive Multi-Core", *in IEEE Transactions on Computers*, vol. 1, doi:10.1109/TC.2013.115.

[10] Parallel Programming: Do you know Pipeline Parallelism?, Web: http://www.futurechips.org/parallel-programming-2/parallel-programming-clarifying-pipeline-parallelism.html

[11] P. Marshall, K. Keahey, T. Freeman "Improving Utilization of Infrastructure Clouds", *in Cluster, Cloud and Grid Computing (CCGrid 2011): Proceedings of the IEEE/ACM Interna-tional Symposium*, Newport Beach, CA, USA, May 23-26, 2011.

[12] E. Tyutlyaeva, E. Kurin, A. Moskovsky, S. Konuhov "Abstract: Using ActiveStorage Con-cept for Seismic Data Processing", *in High Performance Computing,Networking, Storage and Analysis (SCC)*, 2012 SC Companion: 2012, pp. 1389-1390.

[13] M. Iverson, F. Ozguner "Dynamic, Competitive Scheduling of Multiple DAGs in a Distri-buted Heterogeneous Environment", *in Proceedings of Seventh Heterogeneous Computing Workshop*, Orlando, Florida, USA, March 30, 1998, pp. 70-78.

[14] M. Maheswaran, S. Ali "Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems", *in Journal of Parallel and Distributed Computing*, vol. 59, No. 2, pp. 107-131, 1999.

[15] R. Milner, *Communicating and Mobile Systems The Pi Calculus*. Cambridge Unniversity Press, 1999.