Visual Parallel Programming as PaaS Cloud Service with Graph-Symbolic Programming Technology

Darya Egorova <dasharapova@mail.ru>,
Victor Zhidchenko <vzhidchenko@yandex.ru>
Software Systems Department, Information Science Faculty
Samara State Aerospace University (SSAU)
Samara, Russia

Abstract. Most computer programs are created in textual form. From high-level programming languages to CPU instructions both programmer and computer work with sequences of characters and words. Textual representation of the program combines centuries-old tradition of writing as the universal form of fixing human thoughts with ease of interpretation and analysis of text by computer. The sequential nature of text makes it suitable for description of instruction sequences and sequential algorithms. At the same time the text is inconvenient for clear representation of parallel programs. In such programs it is important to depict instructions that can be executed concurrently. In this case the graphical (visual) representation is more suitable.

In this paper we present the visual approach to parallel programming provided by Graph-Symbolic Programming Technology. This technology uses text to represent small sequential subprograms (mathematical expressions or small methods). Visual representation in graph form is used to depict program logic and concurrency. The basics of this technology are considered as well as advantages and disadvantages of visual parallel programming. Synchronization primitives used in Graph-Symbolic Programming Technology and their visual form are described. The method is proposed for compact and clear representation of multiple similar parallel processes.

The technology is being implemented as a PaaS cloud service that provides the tools for creation, validation and execution of parallel programs on cluster systems. The current state of this work is also presented. We argue that visual programming and cloud technologies provide the capability of shared development of programs and algorithms that text programming lacks. The visual programming in such implementation gains the features of the visual modeling.

Keywords: parallel; programming; visual; graph; tool; cluster; cloud

DOI: 10.15514/ISPRAS-2015-27(3)-3

For citation: Egorova Darya, Zhidchenko Victor. Visual Parallel Programming as PaaS Cloud Service with Graph-Symbolic Programming Technology. *Trudy ISP RAN/Proc. ISP RAS,* vol. 27, issue 3, 2015, pp. 47-56. DOI: 10.15514/ISPRAS-2015-27(3)-3.

1. Introduction

Text is traditionally used for describing computer programs. While programs are sequential, it is convenient to express them as text, because the nature of text is sequential. A sequence of letters comprises a word. A sequence of words comprises a sentence. A sequence of sentences forms a text. An order of letters in a word, an order of words in a sentence and an order of sentences in a text are very important. Changing any of them can substantially change the text, especially when this text describes some computer program.

On the other hand, when a program is parallel, its text representation becomes inconvenient. In parallel program you want to see which parts of a program can run concurrently and sequential text form can not show it. You have to imagine interdependencies between different program parts and guess possible combinations of their concurrent execution. When the program is large you have to scroll it up and down to see the parts which actually can run concurrently.

This is where a graphical representation can help. A graphical or visual form is usually bidirectional, so you can easily distinguish sequential and parallel parts of a program. Another important factor is that visual representation is more suitable for human comprehension then a text. When you want to explain something you often get a piece of paper and begin to draw a scheme. The drawing is usually more explanative than a text, it is more compact and is easier to remember.

There is also a substantial disadvantage in using graphics for parallel programs representation. A parallel program often consists of hundreds or thousands of threads or processes and the actual number of them is may be unknown prior to program’s execution. Moreover, the number of threads can vary during execution. When you write such a program in the text, it can be very compact. The clarity still suffers but due to the compactness it is quite easy to imagine the threads structure. Trying to depict such program graphically leads to more complex representation of it. As you can not display thousands of threads on one picture, you have to replace them with some abstract graphics structure. The clarity suffers as well as in the case of the text. So instead of the intuitively clear picture you get some abstraction which is less compact than text and whose usability depends on the chosen abstract form.

There are many ways the visual means are used in programming. Most of them are auxiliary to the "traditional" text programming as they help to perform some particular tasks like building class diagrams, dependency graphs or trace logs. Natural visual programming is provided by visual programming languages. Most of them represent a program as a graph which consists of nodes connected to each other by some links (directed or undirected). Depending on the meaning of nodes and links there are many different approaches to represent a program which can be split into several sets:

- UML diagrams [1]
- Domain-specific Visual Languages
- Petri Nets

- Finite-state and Automata-based Programming [2]
- Data Flow Diagrams
- Control Flow Diagrams

In this paper we describe the present results of the work carried out during several years in Samara State Aerospace University (SSAU) in developing methods and tools for visual parallel programming. We use as a basis the visual programming technology for sequential programming, which is called Graph-Symbolic Programming Technology (GSP-technology) also developed in SSAU [3]. We have extended this technology to describe parallel programs and have evolved it through several desktop versions to development environment working with computing cluster. Today we are working on migrating this technology to the cloud and making PaaS service for visual parallel programming. The results of our work have been used as methods and tools of parallel programming in the education process in SSAU and in research activity in the area of numerical analysis.

2. The Basics of Graph-Symbolic Programming Technology

GSP-technology represents the program as a graph. The nodes of this graph are little programs (modules), which perform simple operations on variables of project domain. The set of variables form a data dictionary.

The nodes are connected with links. The links show the flow of control between the nodes. Every link is provided with the predicate – a logic condition, which permits or denies the flow of control by this link. This condition is a logical function, defined on variables from the data dictionary.

There are situations, when several links going from one node have a true predicate. To resolve this issue, each link has a priority. The link with the highest priority defines the flow of control.

A graph may contain another graph as a node – so, the program is a graph hierarchy. Fig. 1 shows an example graph that solves quadratic equations.

The benefits of GSP are:

- Clear and compact representation of the control flow in a program.
- Elimination of many programming errors as graphic representation is very simple for a human and helps to see many logic errors and inconsistencies.
- Simplicity of the program modification.
- Automatic data flow between the nodes. A programmer is protected from making an error on this stage.
- The program structure is stored into a database. It helps to perform many automatic tasks, such as graph structure verification, measuring of graph complexity, automatic control of graph hierarchy consistency, automatic testing and convenient debugging of programs, automatic creating of program documentation.

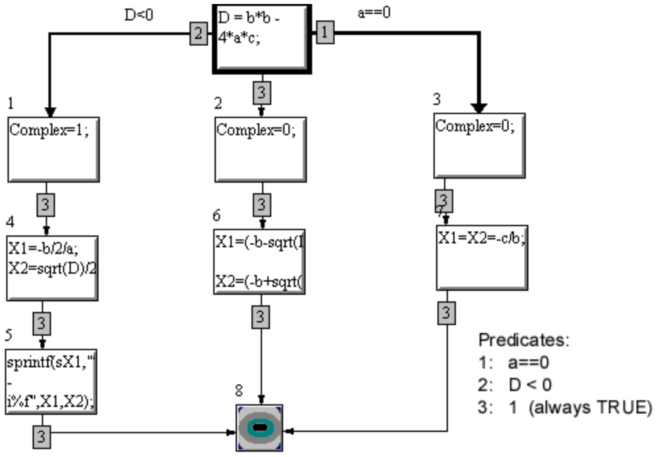

Fig. 1. *Graph of a program for solving quadratic equations*

Being sequential by default, the GSP-technology was further developed for creating parallel programs. GSP graphic representation of programs helps to solve main parallel programming problems:

- Program's visualization.
- Complexity of the interprocess synchronization.

Many tasks have explicit parallelism. The trivial example is determination of real roots of a quadratic equation. GSP graphic representation is very suitable for such tasks. You can simply draw two (or several) parallel branches instead of thinking how to put in order different tasks and how to represent them in a convenient manner.

The graphic language of GSP-technology is expanded with two types of links:

- The parallel link (a link that shows the beginning of a parallel branch) is labeled with the circle in the beginning.
- The terminating link (a link which determines the end of a parallel branch) is labeled with inclined segment.

The program is divided into several processes, which can be performed in parallel. Each process is represented as a separate branch - a set of nodes interconnected with ordinary links and executed sequentially. The number of branches is unlimited. It is forbidden to connect two nodes from different branches.

All branches operate on the same set of data defined in data dictionary. Sometimes, for the purposes of performance optimization and convenience, it is necessary to define local copies of the same data for each parallel branch. It is accomplished by setting the flag "local" for the corresponding variable in data dictionary. The variables with "local" flag set are created in each process separately during execution.

Synchronization is accomplished with a semaphore technique. A special "synchronization graph" is constructed together with the main program graph. The nodes remain unchanged while the links represent nodes interdependences. A link, drawn from $Node_1$ to $Node_2$, means, that $Node_2$'s execution depends on $Node_1$'s state. Transmitting of Nodes' state is made by means of messages.

$L_c = [C^k_{i0,j0}, C^k_{i1,j1}, \dots C^k_{im,jr}]$ is a Message list, where $C^k_{i,j}$ is a message with the number k, sent to $Node_i$ from $Node_j$.

If L_c contains $C_{i,j}$, then $Node_i$ informs $Node_j$ about the finish of its execution.

Every node checks messages addressed to it, before execution. A special semaphore predicate is evaluated on these messages. In accordance with the previous example:

$R_j = f(C^k_{i0,j}, C^k_{i1,j}, \dots, C^k_{im,j})$ is a semaphore predicate of $Node_j$. R_j is a logical function. If R_j = TRUE, then $Node_j$ starts execution, in other case it waits for the truth of R_j.

If all data in a program are independent and there is no need to synchronize parallel branches, the synchronization graph becomes unnecessary and is not built. When it is necessary to synchronize some parts of parallel branches, the user draws synchronization links between the corresponding nodes depicting the sources and targets of synchronization messages. The rest of synchronization graph is implicit and is built automatically.

The process of parallel program development in GSP-technology includes the following steps:

- Data dictionary setup – determining types and variables, needed to solve a problem.
- Modules generation. Modules are written in one of the programming languages (C++ is now supported). They are executed sequentially.
- Drawing the program graph.
- Predicates generation. Predicates are written as boolean functions in the same programming language as modules.
- Drawing the synchronization graph if necessary.
- Semaphore predicates generation for the nodes being synchronized.
- Program compiling and building an executable file.

Fig. 2 shows an example of the graph of the parallel program.

The programming environment of GSP-technology comprises the visual editor for drawing of graphs and defining data and modules, the graph compiler for generating C-source files from graphs and the C-compiler for generating of executable file. Execution environment of GSP-technology uses Message Passing Interface (MPI) for parallel programs execution. Programs generated with GSP-technology can work on clusters and other systems with MPI support.

Each parallel branch is presented with dedicated MPI process.

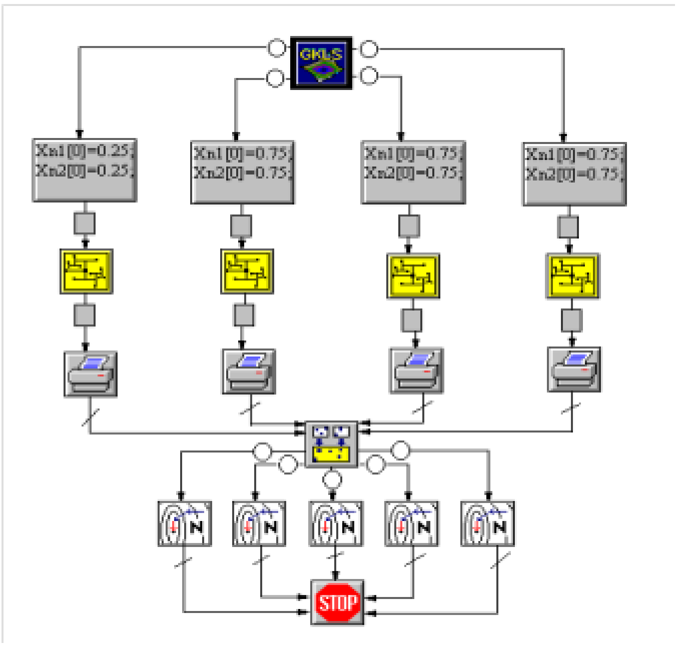

Fig. 2. *Graph of a parralel program for global optimizaion*

To emulate shared memory model in MPI environment, a special memory manager is developed. It allocates memory for data dictionary, initializes program's variables, transmits data to and from the processes and frees unused memory. Memory manager is executed in dedicated MPI process. It is a program that receives data requests from different processes and reads/writes data to or from the memory. Memory manager eliminates memory conflicts between processes.

The parallel program can contain many processes. When there are hundreds or thousands of processes it is inconvenient or just impossible to draw such number of parallel branches on the graph. For such cases GSP-technology uses a special kind of graph nodes called "multitop".

Multitop is represented as one node on the graph and has three parameters associated with it: the module or graph being executed with many processes, the number of parallel processes (branches) represented by the multitop, and the name of the variable which holds the sequence number of each process generated by the multitop. The variable is used within the multitop's module or graph to define its actual function in the same manner as the process rank is used in MPI.

Fig. 3 shows an example of the graph which uses multitops to describe the program similar to that on the Fig. 2 running on 500 processes.

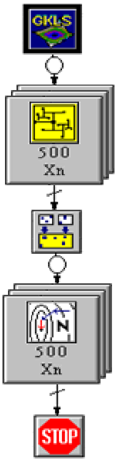

Fig. 3. *Graph of a parralel program for global optimizaion with multitops*

Large number of processes in parallel program is usually used to perform some similar tasks on different independent data without synchronization between the processes. Representation of such tasks as a multitop seems to be a tradeoff between the clarity and the compactness.

3. Present state and future development

For a long time the graph editor in GSP-technology was a desktop application. It comprised graph compiler as a component and was dependent on external C-compiler and database management system (DBMS). This had led to the difficulties in deployment of the system. To install the system in some new location (for example in laboratory classes) one should install the graph editor, then install and properly configure an external C-compiler and DBMS. Using a cluster as a target system for the programs built in GSP-technology requested the direct access to the cluster through the SSH protocol.

To make the use of the GSP-technology easier the web-version of the graph editor was developed. The web-server and DBMS were installed together on the same host and provided remote access to the editor. The editor worked with the database locally and had an SSH connection to the cluster. The main disadvantage of such a system is that the web-interface applies some restrictions to the editor making it less convenient for the users than a desktop application.

Cloud computing has made it possible to combine the rich interface capabilities of desktop graph editor with the centralized management of the hole system for many users. We are working on the development of the Platform as a Service (PaaS) system which will provide visual parallel programming with GSP-technology. PaaS system comprises one virtual machine which hosts the web-server and database and has an SSH connection to the cluster. Many virtual machines can also be run in the same cloud environment each hosting the desktop version of the graph editor. As the database is the same for the web-based and desktop graph editors, it is possible to work on the same project for the team of developers using both versions of editors concurrently.

Some additions have been made to the desktop version of the system. The registration and subsequent authorization of the users running the desktop version was added. During the logon process the user can see the status of other users (online/offline or working with the same project as the current user). All changes made by the user during the session are logged to the database. It is necessary for producing the snapshots - the states of the project development process when some valuable results are achieved, for example, for saving the intermediate working versions of the algorithm which is under development. Another goal of user activity logging is to track the changes made by different users and by the same user in different versions of the system. With logging it is much easier to remember what exactly you have changed while working with the project from the other place (for example, from home) or to understand (and also to explain) the changes made to the graphic model of the program by some other person.

Visual programming can benefit from cloud computing as it provides the capability of shared development that text programming lacks. With text programming the basic tool of team software development is version control system. The concurrent editing of the same file with source code is practically useless. The basic approach is the division of project to smaller tasks, assigning them to different developers and combining results with version control system. With visual programming tool running in the cloud it becomes possible to work on the same graph concurrently. Such shared work is meaningful and can be convenient due to the compactness of visual representation. Editing the same graph concurrently you can easier develop the proper solution of a problem or find the error in a program faster. The visual editing process is similar to the process of discussing something, while graphically illustrating the main ideas being discussed. The visual programming in such implementation gains the features of the visual modeling.

The main issues to resolve in PaaS visual programming service being developed are the following: concurrent work of several users with one project, versioning, compiling and running parallel programs from the desktop virtual machines on the cluster, optimization of the communication between the system and the cluster.

There are also many tasks in the development of the GSP-technology: dynamic processes creation in MPI programs generated by GSP-technology, direct local data exchange between the parallel branches, creation of graph compilers for other parallel programming technologies like OpenMP and CUDA, making interfaces with other programming languages, technologies and libraries in order to leverage code reuse.

References

[1]. H. Gomaa, "Designing Concurrent, Distributed, and Real-Time Applications with UML," Addison Wesley Object Technology Series, Reading MA, 2000.
[2]. N.I. Polikarpova, A.A. Shalyto "Automata-based programming," SPb.: Piter, 2009 [Поликарпова Н.И., Шалыто А.А. Автоматное программирование. СПб:Питер, 2008. – 167 с.]
[3]. A.N. Kovartsev, V.V. Zhidchenko, D.A. Popova-Kovartseva, P.V. Abolmasov "The basics of graph-symbolic programming technology," Proceedings of the Open semantic technologies for intelligent systems (OSTIS-2013) III international conference, pp. 195-204, 2013 [Коварцев, А.Н. "Принципы построения технологии графосимволического программирования" / А.Н. Коварцев, В.В. Жидченко, Д.А. Попова-Коварцева, П.В. Аболмасов // Труды II Международной научно-технической конференции «Открытые семантические технологии проектирования интеллектуальных систем». -2013. - С. 195-204.]

Облачный PaaS-сервис визуального параллельного программирования в технологии графо-символического программирования

Дарья Егорова <dasharapova@mail.ru>,
Виктор Жидченко <vzhidchenko@yandex.ru>
Самарский государственный аэрокосмический универсистет (СГАУ),
443086, Россия, г. Самара, Московское шоссе, 34

Аннотация. Большинство программ создается в текстовом виде. От языков высокого уровня для машинных инструкций программист и компьютер имеют дело с последовательностями символов и слов. Текстовая форма представления программы сочетает в себе многовековые традиции письменности как универсального способа фиксации человеческих мыслей с удобством интерпретации и автоматического анализа текста вычислительным устройством. Последовательная природа текста делает естественным его применение для описания последовательностей инструкций и последовательных алгоритмов. С другой стороны, она препятствует наглядному описанию параллельных программ, когда важно показать не последовательные, а одновременно исполняющиеся инструкции. Для этих целей более удобны графические (визуальные) средства.
В работе представлен визуальный подход к параллельному программированию, реализованный в технологии графо-символического программирования. Технология использует текст для описания небольших последовательных фрагментов программы (математических выражений и

простых подпрограмм). Для наглядного изображения логики программы и для описания параллелизма применяется визуальное представление в виде графа. В статье рассмотрены основы технологии графо-символического программирования, а также преимущества и недостатки визуального параллельного программирования. Приведено описание механизмов синхронизации, используемых в рассматриваемой технологии, а также визуального представления этих механизмов. Предложен способ наглядного изображения большого количества однотипных процессов параллельной программы.

Описано текущее состояние работ по реализации технологии графо-символического программирования в виде облачного PaaS-сервиса, предоставляющего средства для создания, анализа и выполнения параллельных программ для кластерных систем. Показано, что облачные технологии в сочетании с визуальным программированием делают возможным принципиально новый подход к коллективной разработке не только программ, но и алгоритмов, недоступный в традиционном текстовом программировании. Визуальное программирование при этом приобретает свойства визуального моделирования.

Keywords: parallel, programming, visual, graph, tool, cluster, cloud

DOI: 10.15514/ISPRAS-2015-27(3)-3

Для цитирования: Егорова Дарья, Жидченко Виктор. Облачный PaaS-сервис визуального параллельного программирования в технологии графо-символического программирования. Труды ИСП РАН, том 27, вып. 3, 2015 г., стр. 47-56 (на английском языке). DOI: 10.15514/ISPRAS-2015-27(3)-3.

Список литературы

[1]. H. Gomaa, "Designing Concurrent, Distributed, and Real-Time Applications with UML," Addison Wesley Object Technology Series, Reading MA, 2000.

[2]. N.I. Polikarpova, A.A. Shalyto "Automata-based programming," SPb.: Piter, 2009 [Поликарпова Н. И., Шалыто А. А. Автоматное программирование. СПб:Питер, 2008. – 167 с.]

[3]. Коварцев, А.Н. "Принципы построения технологии графосимволического программирования" / А.Н. Коварцев, В.В. Жидченко, Д.А. Попова-Коварцева, П.В. Аболмасов // Труды II Международной научно-технической конференции «Открытые семантические технологии проектирования интеллектуальных систем». -2013. - С. 195-204.