Journal of Computer and Communications, 2017, 5, 129-139
http://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

Programming System PARCS

A. V. Anisimov, A. V. Derevianchenko, P. P. Kuliabko, O. M. Fedorus

Computer Science and Cybernetics Faculty, Taras Shevchenko National University of Kyiv, Kyiv, Ukraine
Email: ava@unicyb.kiev.ua, alexanderder@mail.ru, kpp1@ukr.net

How to cite this paper: Anisimov, A.V., Derevianchenko, A.V., Kuliabko, P.P. and Fedorus, O.M. (2017) Programming System PARCS. *Journal of Computer and Communications*, **5**, 129-139.
https://doi.org/10.4236/jcc.2017.59009

Received: April 19, 2017
Accepted: July 25, 2017
Published: July 28, 2017

Abstract

PARCS (Parallel Asynchronous Recursive Control System) programming tools that allow unified add-on parallel extensions over traditional programming languages are described. The PARCS model is based on the conception of a control space, which is used to describe parallel interacting processes. Structurally, the control space consists of addressable "points" and "channels". Executing modules are assigned to points and communicate through channels connecting points. Recursive embeddings of processes are allowed. The effective implementation of PARCS on cloud platforms Microsoft AZURE and Amazon EC2 is also presented.

Keywords

Parallel Computing, Programming Language, Control Space, Cloud Computing

1. Introduction

Traditional sequential programming languages, operating systems, and computer architectures are mainly conformed to each other. Therefore, the majority of programs get the mobility property. As far as parallel computations are concerned, positive solutions for conformity could be achieved only for a limited number of local areas. In parallel programming, for the same task an algorithm could be effective in one parallel environment and completely inefficient in another one. Meanwhile, the rapid advance of information technologies demands the need to create convenient tools for parallel programming and related technologies. The terms "parallel" and "concurrent" are increasingly used in relation to architecture of computing systems, operating systems, algorithms, programming languages, data structures, and databases. Against this background, the development of descriptive tools well-suited for logical level of algorithmic concurrency is of great importance.

There are two ways for introducing parallelism into programming languages.

The first one is to build-in basic parallel programming features like in languages C++, Java, C#, Python. The second approach relates to constructing special add-ons over standard procedural languages: MPI, OpenMP, Cuda, OpenCL, JavaCL. We propose PARCS (Parallel Asynchronous Recursive Control System) as a universal add-on extension to the base programming languages.

Traditional sequential programming languages, operating systems, and computer architectures are mainly conformed to each other. Therefore, the majority of programs get the mobility property. As far as parallel computations are concerned, positive solutions for conformity could be achieved only for a limited number of local areas. In parallel programming, for the same task an algorithm could be effective in one parallel environment and completely inefficient in another one. Meanwhile, the rapid advance of information technologies demands the need to create convenient tools for parallel programming and related technologies. The terms "parallel" and "concurrent" are increasingly used in relation to architecture of computing systems, operating systems, algorithms, programming languages, data structures, and databases. Against this background, the development of descriptive tools well-suited for logical level of algorithmic concurrency is of great importance.

There are two ways for introducing parallelism into programming languages. The first one is to build-in basic parallel programming features into a base language like in languages C++, Java, C#, Python. The second approach relates to constructing special add-ons over standard procedural languages: MPI, OpenMP, Cuda, OpenCL, JavaCL. We propose PARCS (Parallel Asynchronous Recursive Control System) as a universal add-on extension to the base programming languages.

2. General Introduction to PARCS

The PARCS-conception appeared in early 80-s of the last century as an attempt to create programming tools for computers of non von-Neumann architectures [1]. The key idea behind PARCS is the representation of a complex parallel process as executional activities developing and communicating in some logically connected space. Current activities of a process are assigned to some space coordinates that define communication addresses. We call such a communication structure a Control Space (CS). Structurally CS consists of addressable points and channels. Both points and channels could be constants or variables. Ends of a channel are points, and points are connected by channels. Transmitting data or another information should be done only through channels.

Another conceptual feature of PARCS is an Algorithmic Module (AM), which is a program (procedure) in a base language extended by special commands for interacting with CS. Thus, CS can be viewed as a skeleton of a parallel process carrying executional AMs.

In our constructions of PARCS, we tried to maximally separate commands for CS from specific peculiarities of a basic procedural language. This allows us to create tools to describe universal parallel extensions of programming languages.

Of course, any specific implementation of PARCS depends on a chosen programming platform.

PARCS-constructions allow unified and parameterized parallel extensions of traditional programming languages.

Let *L* be a chosen programming language (PASCAL, C, C++, JAVA, Python, C#, etc.). Basic data types used in PARCS programming are points and communication channels, which connect points. An algorithmic module is a program written in *L* extended by the following commands: SEND DATA TO CHANNEL, ACCEPT DATA FROM CHANNEL, CREATE/DELETE POINT, CREATE/DELETE CHANNEL, ASSIGN AM TO POINT and some others. This way a communication structure is built over *L*. Topologically, CS can be viewed as a dynamically changeable graph. The fulfillment of AM can modify the structure of CS: attaching or deleting points and channels, blocking some processes, creating and destroying AM. Independent AMs could be executed in parallel. Theoretically, CS points can include into itself control subspaces of an arbitrary depth. AMs located in points can interchange by messages through connecting channels. We denote such an extension by PARCS-L.

To date, several PARCS products have been developed: PARCS-PASCAL [2], PARCS-Modula2 [3], PARCS-FORTRAN [4], PARCS-C [4], PARCS-Java [5], PARCS-Cuda [6], PARCS-Python [7], PARCS-C# [8] [9].

3. PARCS Model

Algorithmic models developed on the basis of the PARCS-technology (and using the means of a particular PARCS-programming system) are called "PARCS-models". The development of PARCS-models is implemented at two interrelated levels: logical and program. At the logical level PARCS programming tools allow a user to describe explicitly the resource allocation, all possible commutation and re-commutation connections (with taking into account the recursive algorithm unfolding) or get implicitly the logical structure of the algorithm, resource allocation and re-commutation scheme as a result of the PARCS model functioning. Management, data and rules of interaction between the data and management are described at the program level, with taking into account the corresponding logical structure (it looks like as a "filling" of the logical structure).

The PARCS model is defined as follows:

a) A particular set of basic algorithmic modules (AM) is defined;

b) During the model functioning it is permitted: creating (perhaps recursively) active copies of AM, creating AM with "mutational" changes which are determined by the current situation;

c) The model functioning consists of the creation and destruction of AM active copies, their partially-decentralized asynchronous functioning and dynamic interaction.

Programming PARCS-tools are intended to describe the interaction and recursive-parallel development of the processes which are built on the base of the basic language. Low level PARCS-tools can be configured for single processor

(for simulation) or multiprocessor computers (computer networks, multi-machine complexes, cloud computing, etc.), where message exchange is used to support connections between parallel processes or some other mechanism providing the process synchronization and communication.

The PARCS-environment is provided by the extension of the basic algorithmic language due to the operations with the CS (level of logical structure).

Many well-known abstract models such as Turing machines, recursive functions, program schemes, etc. have been elaborated to deal with sequential algorithms. For parallel programming, algorithmic models strongly depend on a functioning platform. For instance, the same algorithm could demonstrate completely different effectiveness on parallel systems of SIMD, MISD or MIMD (in Flynn's classification).

A PARCS-model reflects general common features of various PARCS-systems. PARCS is intended to extend computations prescribed by a program given in a base language L to parallel execution and interchange. Therefore, a PARCS model has two interrelated levels: logical and program. The logical part is associated with the description of program communicating and bootstrapping facilities. This part strongly relies on CS and its interactions with AMs. At this level, PARCS tools allow a user to explicitly or implicitly, in the case when recursion is used, describe resource allocation, all possible commutation or re-commutation channel connections between algorithmic modules.

The second PARCS level completely depends on a computations model that the base language L uses. Control, data and the interaction rules between control, data and a logical structure are specified at this level.

The PARCS functioning is as follows:

a) A specific set of basic starting AMs is defined;

b) A starting configuration of CS with AMs assigned to certain points is settled;

c) During the execution cycle creating, possibly with the use of recursion, new active copies of AMs can be done;

d) Destruction of AMs can be done;

f) Active AM processes the data according to their control commands specified in the base language L;

g) AMs communicate through channels of CS transmitting both data and control.

Points b), c), and d) are specified by commands of the PARCS logical level.

Further, using the PARCS-model, a chosen base algorithmic language, and an implementation platform the corresponding PARCS-technology could be created.

The huge variety of specific programming languages predetermines a diversity of PARCS-implementations.

4. Control Space Conception

CS is a conception that intends to describe the logical structure of a parallel al-

gorithm (system) and to represent its dynamic changes. CS consists of points and channels that connect points. It may have the hierarchical structure. A new CS, according to CS structural recursion, can be generated from a CS point. Points and channels simulate logical processors (resources) and communication channels for information exchange respectively.

Due to the geometrical structure and communication meaning, CS physical implementation can be used to create specialized parallel processors.

From the object-oriented programming point of view the functioning of CS is performed by objects and operations. In this paper, we use definitions at the level of specifications for abstract data types. The levels of representation and implementation of these data types depend on specific implementations of PARCS programming systems.

Objects such as points and channels are determined by the set of operations presented in Table 1.

CS is a finite (but dynamically extending) graph representing the architecture of parallel computation.

We demonstrate two examples of CS structures ("Radian circle" and "Cube") in terms of PARCS-Java (Figure 1 and Figure 2).

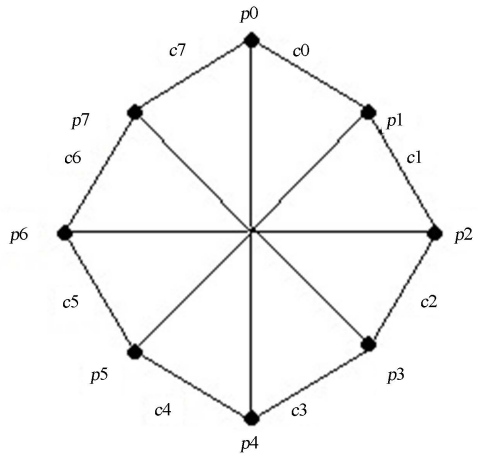

Figure 1. CS-"Radian circle".

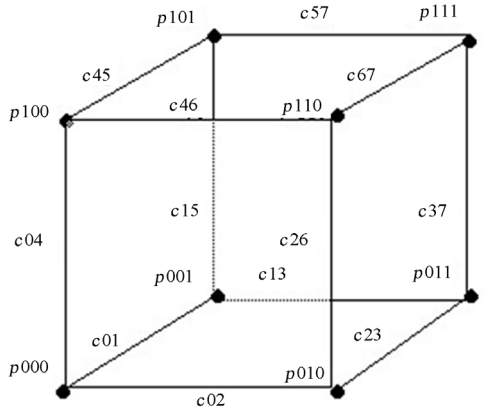

Figure 2. CS-"Cube".

Table 1. Operations on points and channels.

No.	Function	Description
1.	NewPoint()	Creating a new point; returns an identifier of a new point.
2.	NewChannel()	Creating a new channel; returns an identifier of a new channel.
3.	Linkp(p,q,c)	Linking the 2 points p and q with the channel c.
4.	DelPoint(p)	Deleting the point p.
5.	DelChannel(c)	Deleting the channel c.
6.	UnLinkp(c)	Unlinking points which were linked with the channel c; the channel c becomes free.
7.	Howp(p, ar-p[])	Returns the number of points which were linked by channels with the point p; the second parameter is used to return these points.
8.	MyPoint	Returns the point identifier, where this function was called from.

```
// representation of the CS model "RADIAN CIRCLE" Figure 1.
// create Points and Channels-basic elements of the CS.
point[] p = new point[n];
channel[] c = new channel[n];
for (int i=0; i<n; i++) p[i] = curtask.createPoint();
for (int i=0; i<n-1; i++) c[i] = createChannel(p[i], p[i+1]);
c[n-1] = createChannel(p[n-1], p[0]);
// launching relevant AMs;
for (int i=0; i<n; i++) p[i].execute("TASK_AM"+i);

// representation of the CS model "CUBE" Figure 2.
// create Points and Channels-basic elements of the CS.
point[][][] p = new point[2][2][2];
channel[][] c = new channel[8][8];
for (int i=0; i<2; i++);
for (int j=0; j<2; j++);
for (int k=0; k<2; k++);
p[i][j][k] = curtask.createPoint();
for (int i=0; i<2; i++);
for (int j=0; j<2; j++);
for (int k=0; k<2; k++){;
int pi = i*4+j*2+k;
int pj = ((i+1)%2)*4+j*2+k;
if (c[pi][pj] == null);
c[pi][pj] = c[pj][pi] = createChannel(p[i][j][k], p[(i+1)%2][j][k]);
pj= i*4+((j+1)%2)*2+k;
if (c[pi][pj] == null);
c[pi][pj] = c[pj][pi] = createChannel(p[i][j][k], p[i] [(j+1)%2][k]);
pj= i*4+j*2+(k+1)%2;
if (c[pi][pj] == null);
c[pi][pj] = c[pj][pi] = createChannel(p[i][j][k], p[i][j][(k+1)%2]); };
```

```
// launching the relevant AMs;
for (int i=0; i<2; i++);
for (int j=0; j<2; j++);
for (int k=0; k<2; k++);
p[i] [j] [k].execute("TASK_AM"+i+j+k);
```

Using PARCS system we can simulate real computation tasks by constructing different dynamic or static CS configurations (tree, circle, cube etc.).

In parallel computations, one of the most difficult problems is the problem of optimal managing parallel processes. The effectiveness of PARCS systems depends not only on its specific structure but also on its implementation on a specific platform, and on many other external factors such as network or hardware peculiarities.

5. Algorithmic Module (AM)

Another important part of the PARCS-conception is the notion of an algorithmic module. A program in PARCS consists of a sequence of processes. Each process is controlled by a system of AMs assigned to some abstract points of CS. AM sets up a sequential algorithm, either its control or data. Points can be connected by channels, which transmit information. The system of points and channels forms CS of the process and specifies its structural organization. Points and channels are defined as built-in data types. CS can dynamically change its configuration while an algorithm is running. Recursive calls of processes are permitted. Such calls could be initialized by any AM localized inside the calling process.

The call execution launches the process of creating a subspace of CS subordinated to the point, where the call occurs. The corresponding starting AMs are assigned to the points of this subspace.

Inter-module communication operators specify information transmission.

As seen, the PARCS-conception generalizes the conception of communicating sequential processes considered by E. Dijkstra [10] and developed by C.A.R. Hoare [11], P. Brinch Hansen [12].

Conceptually, data transmission over the channels is described by the following set of functions given in **Table 2**.

Table 2. Communication functions of PARCS.

No.	Function	Description
1.	Sendp(c,ms)	Send the message ms to the channel c; returns the success or error code.
2.	Getp(c,ms)	Get the message ms from the channel c; returns the success or error code.
3.	Waitp(L,r)	Wait for the arriving message to the one of the channels from the list L; returns the identifier of the channel. If r = 0 then the message will be deleted else it will be kept.
4.	Delayp(t)	Delay the AM running on time t.
5.	Finish	Finish the AM running.

6. Control Transfer by CS Channels

A control transfer by CS channels is an important and original feature of the CS-conception and associated with it recursive-parallel programming technology. The main purpose of the control transfer is to modify the control part of an active AM-copy as a result of interaction with other active AM-copies. In particular, AM could be assigned to an "empty" point.

In programming languages, such operations have great practical importance for solving a wide class of problems. For example, it is convenient to run the base text of a program as well as several other programs that differ from the basic program in minor modifications.

In traditional programming languages, there exist a number of tools which are particular cases of the control transfer:

- The procedure independent compilation and the transfer of procedure names as an actual parameter;
- Macro processors or preprocessors;
- The text generation of a program and its interpretation in the same program or in another subprogram (such an opportunity is provided, for example, in LISP and in some other interpretative languages). PARCS-commands for the control transfer are given in Table 3.

7. PARCS-System Implementation in Cloud Computing

We have developed and tested procedures of the PARCS deployment on the Amazon EC2 and Microsoft Azure platforms [11] [12]. Herein, we briefly describe the procedure of the PARCS deployment on the Microsoft Azure platform.

We created the project *RestApi* which allows us to obtain information about the current system state and to fulfill operations such as starting a new AM or task cancellation. The project was developed using the *ASP. NET Web API* technology.

It consists of four controllers:

ParcsController calls methods *HostServer API*.

ModuleController—starts a new module with given parameters.

LogController—returns the log-file of the systems work.

AccountController—is responsible for the user authentication.

The main difficulty was to get the remote connection to the first machine and to start deployment of the site *RestApi* on it. To solve this problem, we used the

Table 3. PARCS-commands for the control transfer.

setp(p,AM,pr)	Assign the AM to the point p with priority pr.
setc(c,F)	Send function/procedure F to the channel c; the receiver AM should use function getc(c,F).
setcm(c,ms)	Send text block ms to the channel c; the receiver AM should use function getcm(c,ms).

technique of virtual machines and the *Web Deploy* technology. The last one allows us to download executable files *RestApi* and deploy the site with these files on the web server *IIS*.

Thus, to deploy the site on a virtual machine the following steps should be done:

1. Create a virtual network.
2. Create a virtual machine in the virtual network.
3. *Add IIS Role* in the virtual machine.
4. Install *Web Deploy* in the virtual machine.
5. Set permissions for *Web Deploy* on *IIS*.
6. Post project *RestApi* on the virtual machine by *Web Deploy* using the development environment by *Visual Studio*.
7. Launch *HostServer*.

Afterwards, other virtual machines should be created, the program Daemon should be run on them, and all IP-addresses should be registered in the configuration file HostServer. While creating virtual machines, it is important to join them to a single virtual network that allows them to communicate with each other. Also, while creating a machine, one should configure the endpoints that are ports to access the machine.

Another important feature of this system is its ability to simultaneously perform multiple tasks by different users. If there is no enough capacity the system will wait until processors become free and only then it will run new tasks.

After performing the above action was possible deployment PARCS C# in the cloud Microsoft Azure. We deployed web service system that used a web-interface. As shown in Figure 3, the interface displays the current hosts involved in the calculations and tasks performed. Also present functions add and cancel tasks.

8. Conclusions

The proposed PARCS-system and corresponding PARCS-technologies allow ef-

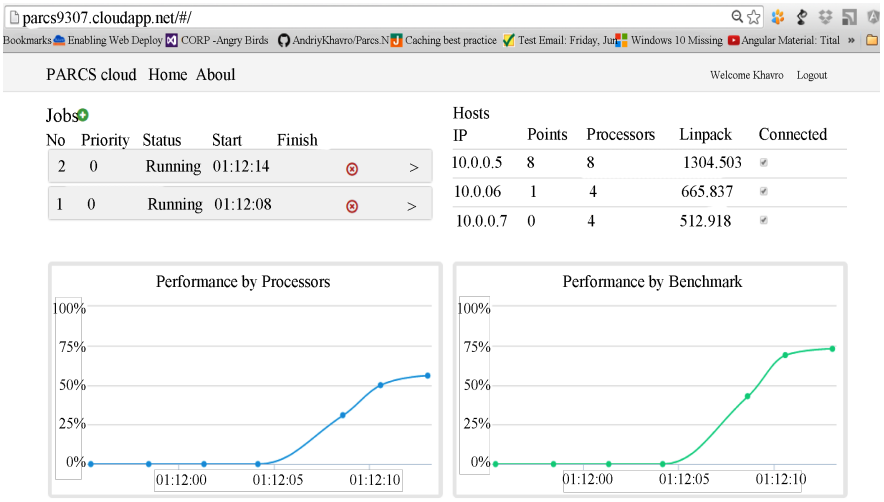

Figure 3. User interface of the PARCS C# system.

fective organization parallel computation processes on different programming platforms. PARCS-tools provide a user with the following possibilities:

- To assist complex computing tasks that require powerful computing and parallel data processing;
- To control information flows in parallel processing systems;
- To accumulate parallel programming algorithms (collecting algorithmic modules) for their further subsequent use.

In the PARCS-technology we mark the following results:

- Efficient tools to control parallel computing processes have been created.
- Practical applications of the PARCS-technology for parallel data processing on computer networks, graphics cards, clusters, and cloud computing have been developed and tested. The PARCS-architectures for cloud computing on Amazon EC2 and Microsoft Azure platform have been developed.
- PARCS-extensions of a number of basic programming languages have been implemented. Corresponding softwares PARCS-PASCAL, PARCS-Modula2, PARCS-FORTRAN, PARCS-C, PARCS-Java, PARCS-Cuda, PARCS-Python, PARCS-C# have been created. They can perform real computing tasks on computer networks, clusters, graphics cards, and cloud technologies.

The effectiveness of algorithms in building different CS models and appropriate parallel programs was experimentally tested.

References

[1] Glushkov, V.M. and Anisimov, A.V. (1980) Controlling Spaces in Asynchronous Parallel Computations. *Cybernetics and Systems Analysis*, **16**, 633-641.

[2] Anisimov, A.V. and Kuliabko, P.P. (1984) Programming of Parallel Processors in Control Spaces. *Cybernetics*, **20**, 404-418. https://doi.org/10.1007/BF01068474

[3] Anisimov, A.V. and Kuliabko, P.P. (1997) Modeling the Petri Nets with PARCS-Tools. *Problems of Programming*, **2**, 45-56. (In Russian)

[4] Anisimov, A.V., Boreisha, U.E. and Kuliabko, P.P. (1991) Programming System PARCS. *Programming*, **6**, 91-102. (In Russian)

[5] Anisimov, A.V. and Derevianchenko, A.V. (2005) The System PARCS-JAVA for Parallel Computations on Computer Networks. *Cybernetics and Systems Analysis*, **41**, 17-26. https://doi.org/10.1007/s10559-005-0037-4

[6] Derevianchenko, A.V. (2011) Application of CUDA in Parallel Computing System PARCS-Java. *Proceedings of the International Conference on Artificial Intelligence, Intelligent Systems AI*-2011, Vol. 1, 62-68. (In Russian)

[7] Anisimov, A.V., Hodovaniuk, M.I. and Kuliabko, P.P. (2016) Parallel Programming in Computer Networks on the Base of PARCS-Technology (Basic Language Is Python). *Bulletin of Taras Shevchenko National University of Kyiv Series Physics & Mathematics*, **3**, 51-54.

[8] Derevianchenko, A.V. and Havro, A.U. (2015) The Use of PARCS.NET and Amazon EC2 for Cloud Computing. *Bulletin of Taras Shevchenko National University of Kyiv Series Physics & Mathematics*, **4**, 111-118.

[9] Derevianchenko, A.V. and Havro, A.U. (2016) Developing Web Service for PARCS.NET and Deploying It in the Cloud Microsoft AZURE. *Proceedings of the 13th International Conference on Theoretical and Applied Aspects of Program Sys-*

tems Development, Kyiv, 81-86.

[10] Dijkstra, E.W. (1968) Cooperating Sequential Processes. In: Hansen, P.B., Ed., *The Origin of Concurrent Programming*, Springer, New York, 65-138. https://doi.org/10.1007/978-1-4757-3472-0_2

[11] Hoare, C.A.R. (1978) Communicating Sequential Processes. *Communications of the ACM*, **21**, 666-677. https://doi.org/10.1145/359576.359585

[12] Brinch Hansen, P. (1978) Distributed Processes: A Concurrent Programming Concept. *Communications of the ACM*, **11**, 934-941. https://doi.org/10.1145/359642.359651