# Development of scalable parallel programs in ParJava environment

Victor Ivannikov, Serguei Gaissaryan, Arutyun Avetisyan, Vartan Padaryan*

Institute for System Programming of RAS,
25 B. Kommunisticheskaya st., Moscow, 109004, Russia

**Key words:** parallel software development, software tools

**Abstract.** In the present paper the ParJava integrated environment supporting the development and maintenance of data parallel Java-programs is discussed. When a parallel program is developed it is necessary to assure not only its correctness, but also its efficiency and scalability. For this purpose it is useful to know some dynamic properties of the program (profiles, traces, slices, etc.). This information may help to modify the program in order to improve its parallel features. ParJava is an integrated environment providing a collection of tools, which help to determine and to improve parallel program properties. These properties are used in symbolic interpreter providing fast execution of SPMD-program allowing to estimate its expected execution time and scalability. It seems to be very convenient facility that allows to estimate limits of scalability and thus to determine needed number of processor nodes.

## 1 Introduction

There is a great deal of interest in using Java for parallel computing. The current research is devoted mainly to MPI interface implementations for Java [1, 2]. ParJava [3] is an integrated environment providing a collection of tools, which help to determine and to improve parallel program properties. It provides the access to various tools that help a programmer to improve his program. It includes the following groups of tools:

- Analyzers, performing static and dynamic analysis of the program being developed;
- The interpreter that provides the symbolic execution of a program to reveal some features of its execution;
- Visualizers mapping the information obtained by analyzers and other tools on the monitor;
- Run-time Monitor, demonstrating the current state of processes.

When a parallel program is developed it is necessary to assure not only its correctness, but also its efficiency and scalability. For this purpose it is useful to know some dynamic properties of the program (profiles, traces, slices, etc.). This information may help to modify the program in order to improve its parallel features. Profiles and traces of the program show the order and the interference of its interprocess communications. The goal of this paper is to describe ParJava tools providing estimation of execution time and scalability of parallel program being developed.

The paper has the following structure. In section 2 the aim of the symbolic execution is discussed. Section 3 presents ParJava tools used to prepare program to symbolic execution. In section 4 we give a brief description of symbolic execution. In section 5 we compare the related systems with corresponding facilities. In section 6 some conclusions are made.

## 2 The aim of the symbolic execution

Detection of dynamic properties of a program requires repeated execution of the program using various sets of initial data, in order to gather statistics. The advantage of symbolic execution against actual execution is that the first one can be performed much faster and using only

*E-mails: {ivan, ssg, arut, vartan}@ispras.ru.

This work is supported by RFBR, grants 02-01-00961, 02-07-90302, 03-07-90198

one processor. Symbolic execution allows to estimate limits of scalability and hence expected execution time for a program during the stage of its development.

We define several models of symbolic execution, which have various precisions in ParJava. Changing execution model we can change the time necessary for symbolic execution. The most precise model uses frequency and time profiles obtained using typical input data, calculated using actual execution of the program on parallel computer system. However, less precise models using profiles obtained by the execution of the program using one processor and model input data are defined. Use of "rough" models allows to reduce program development overheads.

## 3 Preparing program to symbolic execution

To define frequency (time) profile we use the notion of a *step* (we call a step of a program execution of a basic block). The following types of basic blocks are considered: sequence of assignments, condition, function call, and communication function call. Frequency (time) profile of a sequential program is defined as mapping which each step of the program associates with the frequency (time) of its execution. Each profile may be represented as a vector with dimension equal to number of program steps. The profile of the SPMD program is represented by matrix whose lines are profiles of this program's parts executed in parallel.

To obtain time and frequency profiles of a program it should be insrumentated, i.e. calls of instrumental functions should be included in its basic blocks. Instrumental functions should not distort control-flow and data-flow dependences between program's parts executed in parallel. Therefore compensation statements are added.

## 4 Symbolic execution

Symbolic execution is performed by the symbolic interpreter, which uses the control flow graph, data dependences and the profiles. During the symbolic execution of the program its basic blocks that lie on the path defined by the specified input data, are interpreted. Function calls including the calls to communication functions form separate basic blocks. When the basic blocks containing calls to communication functions are executed the apriori information about the communication subsystem is used (according to the model LogGP [4]). This information allows us to determine the duration of transfer of *n* bytes from one node to another. Such information for communication subsystems on the basis of Ethernet and Myrinet is inputted into the system. Symbolic execution allows to estimate program execution time for each number of cluster nodes as well as maximal number of nodes for which scalability is hold.

## 5 Related Works

There are two directions in MPI implementations for Java: native MPI bindings where MPI native library is called by Java programs through Java wrappers [1], and pure Java implementations [2].

As for integrated environments all related works (research and commercial works) are connected with the development of such environments for Fortran and C languages. We may emphasize such freeware environments as AIMS (NASA Advanced Supercomputing Division) [5], TAU (University of Oregon) [6], Pablo (University of Illinois) [7] and commercial systems Vampir (Pallas GmbH) [8], PGPROF (The Portland Group™ Compiler Technology) [9] and others. All these systems employ post-mortem analysis of the program. They provide wide opportunities for trace viewing, gathering of parallel applications profiles and operating with statistics. The analogical tools are implemented in the ParJava environment.

AIMS is the environment most similar to ParJava. It provides the symbolic execution of the SPMD-program. But during the instrumenting of the program in AIMS, unlike in ParJava, the quantity and position optimization of calls to instrumental functions is not executed. The interpreting program is presented internally in AIMS in BDL language in which the presentation of the program is compiled from the blocks of three kinds: "cycle", "communication" and "sequence". In the ParJava environment the program is presented by the control flow graph that supplies a more adequate interpretation. Moreover, in the ParJava environment the mechanism that gives an opportunity to control the process of the symbolic execution is realized. It is also possible to gather the trace of the symbolically executed program, to monitor the states of the program parts executed in parallel and to determine the deadlock automatically.

## 6 Conclusion

The ParJava environment allows to develop portable scalable SPMD Java-programs, providing the application programmer with tools that help him to improve his parallel Java-program. The ParJava environment is on the stage of evolution. The new instrumental tools are being developed (the automatic program parallelizer, the relative debugger of SPMD-programs and others), the problem of performance growth of parallel Java programs is under investigation, the possibilities to apply the Java environment during the preparation of programs for GRID are also studied.

## References

1. S. Mintchev. Writing Programs in JavaMPI. //TR MAN-CSPE-02, Univ. of Westminster, London, UK, 1997.
2. Tong WeiQin, Ye Hua, Yao WenSheng. PJMPI: pure Java implementation of MPI. //Proc. of the 4th Int. Conf. on HPC in the Asia-Pacific Region, 2000.
3. Avetisyan, S. Gaissaryan, O. Samovarov. Extension of Java Environment by Facilities Supporting Development of SPMD Java-programs. //V. Malyshkin (Ed.): PaCT 2001, LNCS 2127, pp. 175 – 180.
4. R. Martin, A. Vahdat, D. Culler, T. Anderson. The Effects of Latency, Overhead and Bandwidth in a Cluster of Workstations. //In Proc. 24th Int. Symp. on Com. Arch. (ISCA'97), June 2 - 4, 1997, pp 85 – 97.
5. J. C. Yan and S. R. Sarukkai. Analyzing Parallel Program Performance Using Normalized Performance Indices and Trace Transformation Techniques. //Parallel Computing. Vol. 22, No. 9, November 1996. pages 1215-1237
6. S. Shende, A. D. Malony, J. Cuny, K. Lindlan, P. Beckman and S. Karmesin. Portable Profiling and Tracing for Parallel Scientific Applications using C++. //Proc. SPDT'98: ACM SIGMETRICS Symp. on Parallel and Distributed Tools, pp. 134-145, Aug. 1998.
7. Luiz DeRose, Mario Pantano, Jeffery S. Vetter, and Daniel A. Reed. Performance Issues in Parallel Processing Systems. /Performance Evaluation: Origins and Directions, Guenter Haring, Christoph Lindemann, and Martin Reiser (eds.), 1999, pp. 133-150.
8. W. E. Nagel, A. Arnold, M. Weber, H.-C. Hoppe, and K. Solchenbach. VAMPIR: Visualization and analysis of MPI resources. //Supercomputer, 12(1): 69--80, Jan. 1996.
9. Bryan Carpenter, Guansong Zhang, Geoffrey Fox, Xiaoming Li, Xinying Li, and Yuhong Wen. Towards a Java environment for SPMD programming. //In David Pritchard and Jeff Reeve, editors, 4th International Europar Conference, volume 1470 of LNCS, 1998.