High productive programming for cluster systems using ParJava environment

Victor P. Ivannikov
Arutyun I. Avetisyan
Sergey S. Gaissaryan
Varvara V. Babkova,

Institute for System Programming RAS
Moscow, Russia
e-mail: ivan, arut, ssg, barbara@ispras.ru

ABSTRACT

This paper is devoted to presentation of ParJava environment being developed in ISP RAS. It provides to application programmer a set of tools supporting design of parallel programs for high performance clusters using Java extended by a standard MPI library.

ParJava allows to make the most part of development using instrumental computer. It is possible because of using the original model of a parallel Java program. The model is designed in such a way that its interpretation allows to predict program execution time (performance), or its other dynamic attributes (profiles, traces, etc.) for given cluster. Accuracy of prediction may be improved using special benchmarks to obtain cluster's parameters and measuring the execution time of basic blocks using single node of the cluster. Using of the interpreter in real applications development (a program of tornado modeling and a program of modeling) demonstrated that prediction error does not exceed 10%.

Use of an instrumental computer instead of the cluster together with new generation language Java appreciably increases the productivity of development.

In closing current research activities devoted to methods and tools allowing to discover program parallelism, automatic parallel code generation and automatic tuning of code to given hardware are discussed.

Keywords

Parallel programming, scientific computations.

1. INTRODUCTION

Evolution of computer and network techniques resulted in parallelism on all levels being the main property of modern computing systems. The cluster systems (distributed memory) with thousands of processors are widely used. Wide production of general purpose multicore processors has begun. Regardless of the fact that up-to-date multicore processors have not more than 16 cores, produces already seriously promise several hundreds and even thousands cores [1]. Moreover, special-purpose processors, holding hundreds of cores on a chip are already produced (graphical accelerators produced by AMD and nVidia). High performance, low power consumption and low cost of special-purpose multicore (usually, these processors are oriented to computer games) led to simulated the trend to use them not only their direct destination. Efforts for wide application of heterogeneous architectures including a general purpose processor together with a set of special-purpose multicore processors (accelerators) to solve general purpose computing tasks. An accelerator has access to both – to its own storage and to common storage of heterogeneous system as well. As examples of such architectures we may consider IBM Cell architecture, architectures using graphical accelerators produced by AMD and nVidia, multicore graphical accelerator Larrabee of Intel.

A nagging problem arose – to design parallel programming languages to provide high productivity of programmers developing parallel applications. However the languages designed in the nineties (HPF [2], UPC [3] etc.) did not solve the problem [4]. As the result programmers performing the industrial design of parallel applications are forced to use so called "assembly level" using a sequential programming language (C/C++, Fortran), designed in 60-70ies, and explicit calls to communication library MPI (for systems with distributed memory), explicit OpenMP pragmas (for systems with common memory), or using CUDA [5] technology (CUDA is a superset of C for Nvidia accelerators). Listed facilities exactly correspond to hardware architecture allowing to develop efficient parallel programs, but expect the high level of comprehension of hardware details.

Thus, the parallel programming presently needs manual tuning of a programs (data distribution, communication patterns, synchronization of access to critical data, etc.). This leads to considerable resource spending and needs high professional skill of application programmers. The cost needed to achieve a good performance and needed degree of scalability often is found too high. Therefore the goal of modern research is the fundamental problem of high productivity [6] of the development of parallel applications, when a sufficient level of performance and scalability is achieved spending acceptable level of development costs. This is especially important as the parallel programming becomes prevalent.

There are many directions of research: the properties of various applications are studied, the efforts to classify applications are made (including to discover some common kernels), the properties of hardware are studied in order to maximize their usage and evolution, the wide spectrum of programming facilities is being researched and designed.

One of directions of research is design of the new generation of parallel programming languages (X10 [7], Chapel [8], Fortress [9], Cilk [10], Brook+ [11] and others). In spite of these works are guided by the past experience they did not yet bring to the success, first of all owing to insufficient level of existing compiler technologies.

Both industrial and research systems supporting the tuning process of programs developed on "assembly level" are implemented. Presently several such systems are known: DDT [12], and TotalView [13] debuggers, TAU [14] tuning facility of Oregon University, etc.

ParJava environment [15], developed in ISP RAS, is one of such facilities. ParJava provides to application programmer a tool-kit supporting development of parallel programs for computing systems with distributed memory (high performance clusters) using Java language extended with a standard message passing library (MPI).

Presently Java environment is of considerable interest in the aspect of high performance computations. It is due to positive properties of Java in aspect of application programming (portability, debugging simplicity, etc.), and to simplicity of development of various tools using Java infrastructure as well. It is necessary to mention such systems as ProActive Parallel Suite [16] (INRIA), MPJ Express [17] (University of Reading and University of Southampton), Distributed Parallel Programming Environment for Java (IBM) etc. Furthermore, the support for Java + MPI has been added to well-known parallel programs development system for C/C++ and Fortran 77/90 TAU.

The following two tasks were studied in ParJava Project: (1) to provide the efficient execution of parallel Java-programs with explicit calls of MPI using high performance cluster systems, and (2) to create a workflow of a parallel program implementation providing the realization of the most part of workflow using the instrumental computer.

In present paper the main features of ParJava environment are described.

2. THE MODEL OF AN SPMD PROGRAM IN THE PARJAVA ENVIRONMENT.

Syntactically the model of an SPMD program is a set of models of all its classes, the model of a class is a set of models of all its methods together with the model of an additional method describing the class fields, its static variables, and their initialization. The model of a method (function) consists of its symbol table and its control tree (in the case of Java it is isomorphic to the abstract syntax tree of the method: the internal nodes of the model correspond to Java control statements, and the leaf nodes correspond to basic blocks. We consider as basic block not only linear sequences of computations (computational basic block) but also calls of methods and functions, and calls of communication functions. The model of a basic block is a tuple containing the descriptor and the body of the basic block (a sequence of expressions or function call). To enable interpretation of a model by parts, the concept of a reduced block is introduced: a reduced block by definition is a dummy basic block with empty body, its descriptor provides the values of dynamic attributes of a basic block or a region being represented by the reduced block (region is defined as a part of a program corresponding to control tree's node).

Each MPI process of the SPMD program is represented in its model by a logical process. A logical process is defined as a sequence of actions (such as execution of a basic block, execution of a data exchange operation, and the like), each action being described by its duration. The durations of computational basic block are measured on the target platform in advance. Each logical process has a model clock. The initial value of model clock of each process is zero. Upon interpretation of each action, its duration is added to the model clock of corresponding logical process.

To reduce the interpretation time and enable one to perform the interpretation on the development computer, only the control flows of the processes and data exchange operations are modeled. This is possible because the execution time (duration) and other dynamic attributes of basic blocks are determined on the target system before the interpretation is started. The interpretation only propagates values of the dynamic attributes to other nodes of the model. Such an approach makes it possible to remove the variables whose values do not affect the control flow of the program from the basic block models. As a result, the part of the computations in which these variables are defined and used becomes dead code and is also removed. As a result, the amount of data to be processed and the total amount of computations performed in the course of the interpretation is reduced appreciably. The procedure just described can sometimes eliminate all the code from some basic blocks; such basic blocks are replaced by reduced blocks.

3. INTERPRETATION OF THE MODEL

The interpretation of an SPMD program model executed on p nodes of a parallel computing system (cluster) is performed in p logical processes each of which is executed in an individual thread. The interpretation consists in computation of control statements to determine a basic block which must be executed next and processing of chosen basic block. After the basic block is chosen the following actions are performed depending on its type:

- For a computational block (the duration d of such blocks is determined in advance on the target platform), the duration d is added to model clock of corresponding logical process, and the control is returned to the interpreter.
- For a reduced block (the duration d and other dynamic attributes of such blocks are determined before it is reduced), the duration d is added to model clock of corresponding logical process, and the control is returned to the interpreter.
- For a block representing a communication function call, the control is returned to the interpreter, where the model of corresponding communication function is executed. In addition to transferring data between logical processes, the model estimates the communication execution time (duration) and adds estimated duration to model clock of corresponding logical process.
- For a block representing a user function call, the control is returned to the interpreter, where the model of the function (method) is executed. In the course of the computation model execution, the current state is saved in a stack and the required data structures are loaded. After the interpretation of the model of the function is completed, the state that existed at the moment of calling the function is popped from the stack, and interpretation of the caller function's model is continued.

4.REDUCTION AND INTERPRETATION BY PARTS

There is a reduction operation defined on the elements of the method's model. The reduction of a basic block is the replacement of this block with a reduced block having an empty body and a fixed value of the dynamic attributes. The reduction of an internal node representing a Java statement consists in the depth-first traversal of the corresponding subtree with the sequential reduction of all its descendants beginning with the basic blocks. A necessary condition for a node reducibility is its closeness with respect to control (in the subtree rooted at this node, no values of the control variables used in other parts of the program are evaluated) and its closeness with respect to communications (if the subtree rooted at this node contains a call of a message send (receive) function, the corresponding call of the message receive (send) function must also be contained in the same subtree.

The verification of the closeness property with respect to communications is reduced to finding the pair for each receive (send) operation, i.e., to finding the pairs send–recv. Some such pairs can be found using the static analysis in the course of the model generation; the information about the detected pairs is saved in the model. However, not all the pairs can be found using the static analysis. The remaining pairs are found by dynamic analysis in the course of the model interpretation. The pairs are found using the system predicate Match, which checks if the parameters of the functions send and recv (communicators, tags, and process identifiers) match each other. If a node does not meet the reducibility conditions, one can use the Model Analyzer tool included in ParJava to find the minimal subtree containing this node and satisfying the reducibility conditions.

5. PARJAVA ENVIRONMENT

The interpreter of the model of SPMD program is one of the tools of ParJava environment supporting the process of development of SPMD Java-programs using mainly a source computer. The information about dynamic attributes of an SPMD program obtained using ParJava interpreter allows to estimate its scalability domain walls, and helps application programmer to improve manually his program checking the improvements made by the interpreter.

ParJava environment is integrated with Eclipse [18], the open source Java program development environment, thus forming the unified SPMD-program development environment including both ParJava tools and Eclipse tools (text editor, project build subsystem, incremental compilation facility, etc.).

The first problem which had to be solved during implementation of ParJava was effective implementation of standard MPI library for Java. Several implementations of Java MPI [19, 20] available through Internet do not provide effective data communications and miss some MPI functions. Therefore a new version of MPI for Java environment (mpiJava library) was implemented. The current version of mpiJava supports all functions of MPI-1.1 standard and parallel input/output of MPI-2 standard as well. mpiJava library is implemented using JNI binding techniques by analogy with C++ binding described in MPI-2 standard.

Besides the model builder and its interpreter ParJava provides several tools (such as Loop Analyzer, Model Explorer, Model Editor, etc.) facilitating development and tuning of SPMD Java-programs. ParJava interpreter allows to obtain estimations of speed-up of a program which are precise enough; the error does not exceed 10% on real applications and is estimated as 5% on average. The final values of parameters of the SPMD program can be improved using the target cluster.

6. EXAMPLES AND NUMERICAL RESULTS

Admissibility of practical application of ParJava environment was tested using several sample SPMD programs and one real SPMD program. Here some examples are presented.

The first sample program is very simple. It provides the solution of the system of 10,000 linear algebraic equations by means of Jacoby iteration algorithm (Jacoby sample program). As it is seen on Fig. 1 shows the values of speed up obtained using interpreter (broken curve) are close to those obtained during real execution of the program (firm curve).

The second sample program provides the solution of linear heat conduction equation in rectangular region (Heat conduction sample program). The results are presented in Table 1 (the relative error of estimation did not exceed 0.7%). In this example the number of matrix elements is increased together with number of MPI processes in such a way that

Table 1. The results for Heat conduction sample program (estimated and real execution times)

Number of MPI processes	Execution time (seconds)			
	Estimated (ParJava)	Java 1.4.2 (IBM)	C (ICC7.0)	C (gcc 3.3.3)
16	337,32	353,79	357,51	352,20
32	342,56	359,63	360,20	371,65
48	347,79	369,70	372,42	378,22
64	357,93	370,89	387,12	374,80
72	355,64	378,05	386,00	383,89
80	358,26	375,75	386,78	382,64
96	363,49	388,04	385,68	390,12
104	376,24	400,49	390,32	385,82
120	375,60	391,53	393,64	396,38
128	373,96	394,51	394,73	395,85

amount of data in each process remains practically unchanged (J. Gustafson's scaled-size scalability model [21]).

Understanding that sample programs, being very simple and taking too little time for execution, do not let to appreciate the advantages of ParJava, we added the third example.

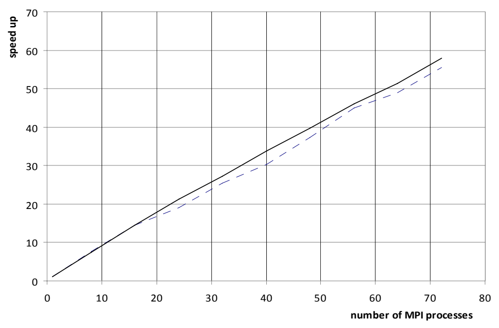

Fig. 1. Estimated and real speed up for Jacoby sample program

The example presents a real scalable parallel program calculating numerical solution of non-linear system of partial differential equations, modeling the processes and origin conditions of intensive atmospheric vortices in 3D compressible atmosphere according to the theory of mesovortice turbulence by V. Nikolaevskiy. The program was developed in the Institute for System Programming RAS in collaboration with the Institute of Physics of the Earth RAS using ParJava environment and is intended to be executed using high-performance clusters. The system of equations was

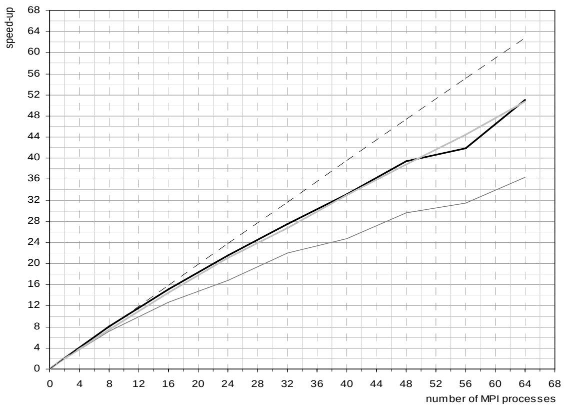

Fig. 2. Comparison of modeled and real program speed-up for blocking and non-blocking communications.

obtained in [22, 23] and is a strongly non-linear system of the mixed type.

The test computations held at the ISP RAS cluster (16 nodes, 4 processor per node, Intel® Xeon® CPU X5355 @ 2.66GHz Myrinet Express 2000).

Program has been modeled by interpreter. The interpretation was carried in several sessions and included partial interpretation of the main loop and some of methods. Each session did not take time more than three hours. The execution time of the program is about 30 hours (it was divided into two intervals using control point mechanism implemented in ParJava environment). The results are presented on Fig.2. Comparison of real and model speed up curves shows that modeling is precise enough: the error did not exceed 5% in average.

Also from Fig 2 is seen, that replacement of blocking MPI communication functions (send, resv) by non-blocking ones (isend, iresv) significantly improves program's speed-up.

7.DETAILS

7.1. Contact adress

Institute for System Programming of the Russian Academy of Sciences,
25, B.Kommunisticheskaya st., Moscow, 109004, Russia
E-mail: arut@ispras.ru
Phone: +(7-495) 912-07-54
Fax: +(7-495) 912-15-24

8. ACKNOWLEDGEMENT

The authors are grateful to V.N. Nikolaevsky, V.A.Padaryan and S.A. Arsenyev for useful critics, discussions and support.

REFERENCES

[1]. David A. Patterson et al. The Parallel Computing Laboratory at U.C. Berkeley: A Research Agenda Based on the Berkeley View. Technical Report No. UCB/EECS-2008-23 http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-23.html. March 21, 2008.

[2]. High Performance Fortran Language Specification, Version 2,0.//High Performance Fortran Forum January 31,1997 http://www.netlib.org/hpf/index.html

[3]. W. Chen, C. Iancu, K. Yelick. Communication Optimizations for Fine-grained UPC Applications. //14th International Conference on Parallel Architectures and Compilation Techniques (PACT), 2005.

[4]. K. Kennedy, C. Koelbel, H. Zima. The Rise and Fall of High Performance Fortran: An Historical Object Lesson // HOPL III: Proceedings of the third ACM SIGPLAN conference on History of programming,2007, San Diego, California, June 09 - 10, 2007, pp. 7-1 – 7-22

[5]. CUDA, Parallel Programming Environment for GPU. http://www.nvidia.com/object/cuda_home.html (In Russian)

[6]. The DARPA High Productivity Computing Systems. http://www.highproductivity.org/

[7]. K. Ebcioglu, V. Saraswat, V. Sarkar. X10: an Experimental Language for High Productivity Programming of Scalable Systems // Proceedings of the Second Workshop on Productivity and Performance in High-End Computing (PPHEC-05) Feb 13, 2005, San Francisco, USA pp. 45-52

[8]. B.L. Chamberlain, D. Callahan, H.P. Zima. Parallel Programmability and the Chapel Language // International Journal of High Performance Computing Applications, August 2007, 21(3): 291-312.

[9]. E. Allen, D. Chase, J. Hallett et al The Fortress Language Specification (Version 1.0) / cSun Microsystems, Inc., March 31, 2008 (262 pages)

[10]. Cilk++ Solution Overview. http://www.cilk.com/multicore-products/cilk-solution-overview/

[11]. Brook+ Streaming Compiler. http://ati.amd.com/technology/streamcomputing/sdkdwnld.html

[12]. Parallel Debugger: DDT. http://www.nottingham.ac.uk/hpc/html/docs/numerical/parallel_ddt.php

[13]. TotalView. http://www.totalviewtech.com/

[14]. Sameer S. Shende, Allen D. Malony. The TAU Parallel Performance System // The International Journal of High Performance Computing Applications,Volume 20, No. 2, Summer 2006, pp. 287–311

[15]. V.P.Ivannikov, A.I.Avetisyan, S.S.Gaissaryan, V.A.Padaryan. Parallel Program Dinamic Characteristic Evaluation Using Models. // Programming and Computer Software, 2006, №4 pp. 21–37

[16]. Brian Amedro, Vladimir Bodnartchouk, Denis Caromel, Christian Delbé, Fabrice Huet, Guillermo L. Taboada. Current State of Java for HPC. Technical report N° 0353. August 2008.

[17]. Mark Baker, Bryan Carpenter, and Aamir Shafi. MPJ Express: Towards Thread Safe Java HPC, Submitted to the IEEE International Conference on Cluster Computing (Cluster 2006), Barcelona, Spain, 25-28 September, 2006.

[18]. http://www.eclipse.org/

[19]. mpiJava Home Page// http://aspen.ucs.indiana.edu/pss/HPJava/mpiJava.html

[20]. Mark Baker, Bryan Carpenter, Geoffrey Fox , Sung Hoon Ko, and Xinying Li. mpiJava: A Java MPI Interface// http://acet.rdg.ac.uk/~mab/Papers/EuroPar-98/, 1998

[21]. J.L. Gustafson, Reevaluating Amdahl's Law// Communications of the ACM, 1988, Vol. 31, No 5.

[22]. Nikolaevskiy V.N. Angular Momentum in Geophysical Turbulence: Continuum. Spatial Averaging Method. Dordrecht: Kluwer (Springer). (2003) 245

[23]. Arsenyev S.A., Gubar A.Yu., Nikolaevskiy V.N. Self-Organization of Tornado and Hurricanes in Atmospheric Currents with Meso-Scale Eddies. // Doclady Earth Science. Vol.396. N.4. (2004) 588-593